Distributed Scheduling of Multiple Bag-Of-Task Applications: Proportionnal Fairness

R. Vannier, A. Legrand

Laboratoire ID-IMAG, CNRS-INRIA Grenoble, France

ALPAGE Working Group, October 19, 2006

Motivation

Large-scale distributed platforms result from the collaboration of many users:

- sharing resources amongst users should somehow be fair.
- Task regularity (SETI@home, BOINC, ...) $\rightsquigarrow$ steady-state scheduling.

Designing a *Fair* and Distributed scheduling algorithm for this framework.

Motivation

Large-scale distributed platforms result from the collaboration of many users:

- sharing resources amongst users should somehow be fair.
- Task regularity (SETI@home, BOINC, ...) $\rightsquigarrow$ steady-state scheduling.

Designing a *Fair* and Distributed scheduling algorithm for this framework.

Motivation

Large-scale distributed platforms result from the collaboration of many users:

- sharing resources amongst users should somehow be fair.
- Task regularity (SETI@home, BOINC, ...) $\rightsquigarrow$ steady-state scheduling.

Designing a *Fair* and Distributed scheduling algorithm for this framework.

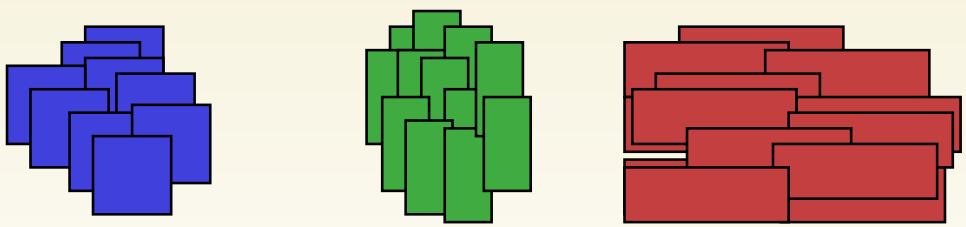
Plan

Plan

Application model

Multiple applications:

- A set A of K applications $A_1, \ldots, A_K$
- Each consisting in a large number of same-size independent tasks $\rightsquigarrow$ each application is defined by a communication cost $w^{(k)}$ (in MFlops) and a communication cost $b^{(k)}$ (in MB).
- Different communication and computation demands for different applications

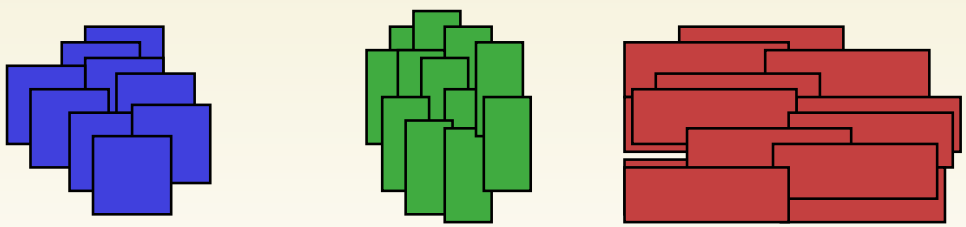


A_1 A_2 A_3

Application model

Multiple applications:

- A set A of K applications $A_1, \ldots, A_K$
- Each consisting in a large number of same-size independent tasks $\rightsquigarrow$ each application is defined by a communication cost $w^{(k)}$ (in MFlops) and a communication cost $b^{(k)}$ (in MB).
- Different communication and computation demands for different applications

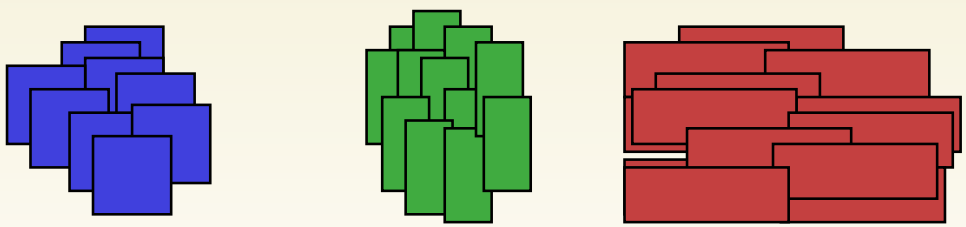


A_1 A_2 A_3

Application model

Multiple applications:

- A set A of K applications $A_1, \ldots, A_K$
- Each consisting in a large number of same-size independent tasks $\rightsquigarrow$ each application is defined by a communication cost $w^{(k)}$ (in MFlops) and a communication cost $b^{(k)}$ (in MB).
- Different communication and computation demands for different applications



A_1 A_2 A_3

Steady-state scheduling

- All tasks of a given application are identical and independant
  $\rightsquigarrow$ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems)
- We only need to focus on average values in steady-state.
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor i per time unit: $\alpha_i^{(k)}$
  - Throughput of application k : $D_k = \sum_{i \in P} \alpha_i^{(k)}$

Steady-state scheduling

- All tasks of a given application are identical and independant
  $\rightsquigarrow$ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems)
- We only need to focus on average values in steady-state.
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor i per time unit: $\alpha_i^{(k)}$
  - Throughput of application k : $D_k = \sum_{i \in P} \alpha_i^{(k)}$

Steady-state scheduling

- All tasks of a given application are identical and independant
  $\rightsquigarrow$ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems)
- We only need to focus on average values in steady-state.
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor i per time unit: $\alpha_i^{(k)}$
  - Throughput of application k : $D_k = \sum_{i \in P} \alpha_i^{(k)}$

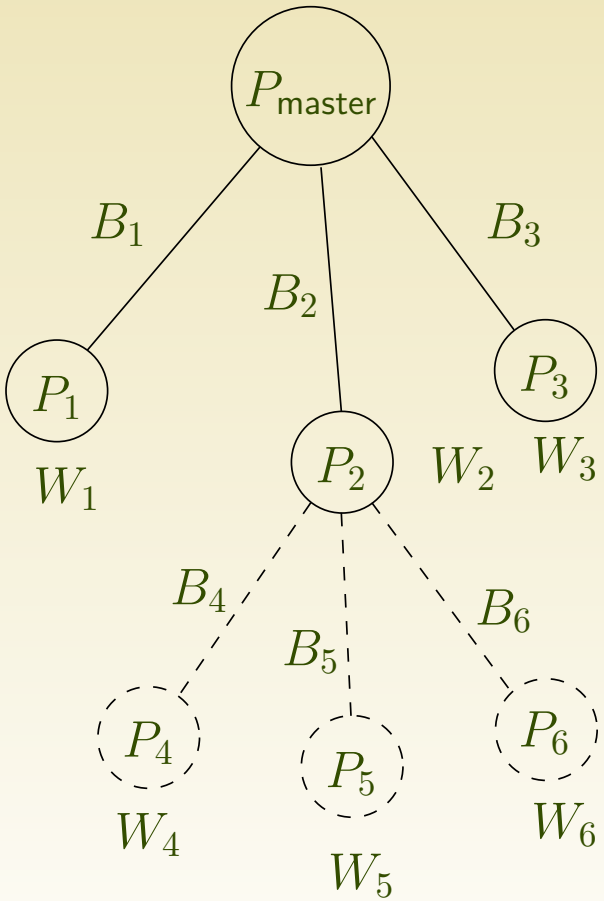
Steady-state scheduling

- All tasks of a given application are identical and independant
  $\rightsquigarrow$ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems)
- We only need to focus on average values in steady-state.
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor i per time unit: $\alpha_i^{(k)}$
  - Throughput of application k : $D_k = \sum_{i \in P} \alpha_i^{(k)}$

Steady-state scheduling

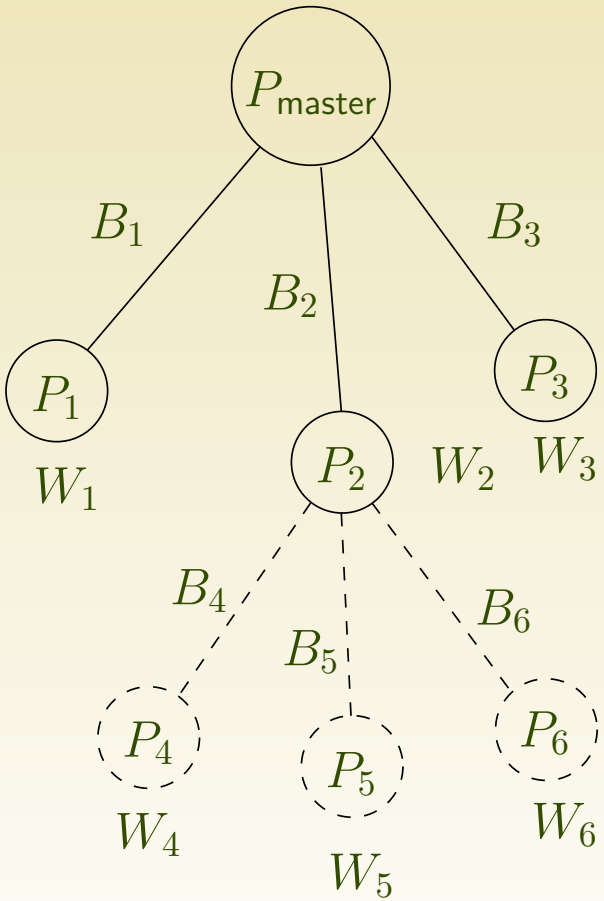
- All tasks of a given application are identical and independant
  $\leadsto$ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems)
- We only need to focus on average values in steady-state.
- Steady-state values:
  - Variables: average number of tasks of type k processed by processor i per time unit: $\alpha_i^{(k)}$
  - Throughput of application k : $D_k = \sum_{i \in P} \alpha_i^{(k)}$

Platform Model



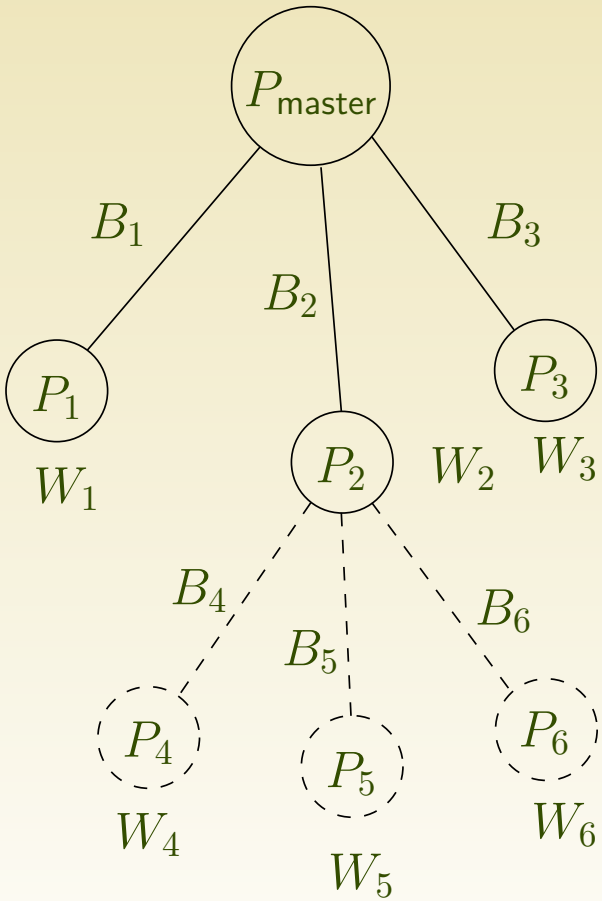
- Target platform: master-worker
  - Master P_{master} holds all tasks initially
  - Workers: $P_1, \ldots, P_p$
  - Father of P_u: $P_{f(u)}$
- Bandwidth of $P_{f(u)} \rightarrow P_u$: B_u (in MB/s)
- Speed of P_u: W_u (in MFlops/s)
- Communications and computations can be overlapped
- Multi-port communication model

Platform Model



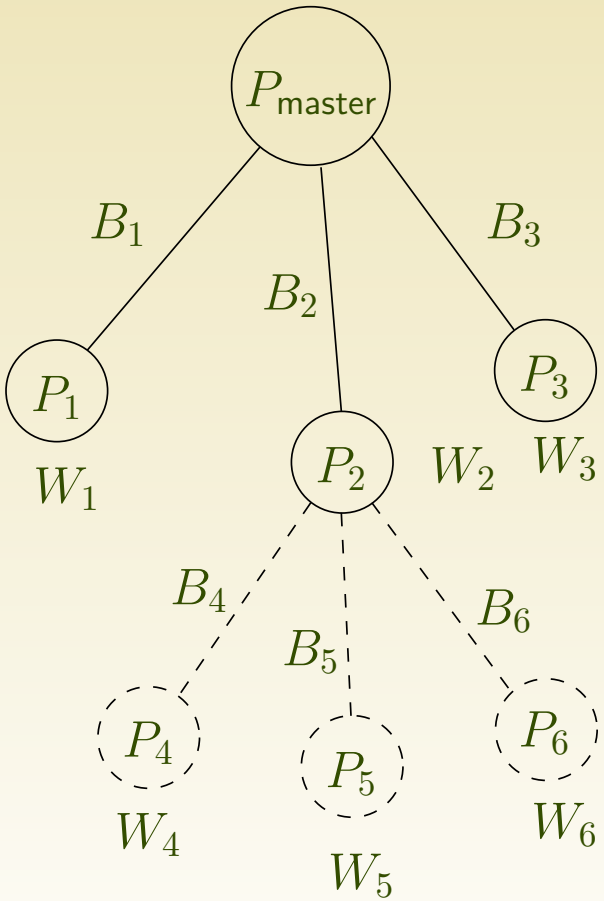
- Target platform: master-worker
  - Master P_{master} holds all tasks initially
  - Workers: $P_1, \ldots, P_p$
  - Father of P_u: $P_{f(u)}$
- Bandwidth of $P_{f(u)} \rightarrow P_u$: B_u (in MB/s)
- Speed of P_u: W_u (in MFlops/s)
- Communications and computations can be overlapped
- Multi-port communication model

Platform Model



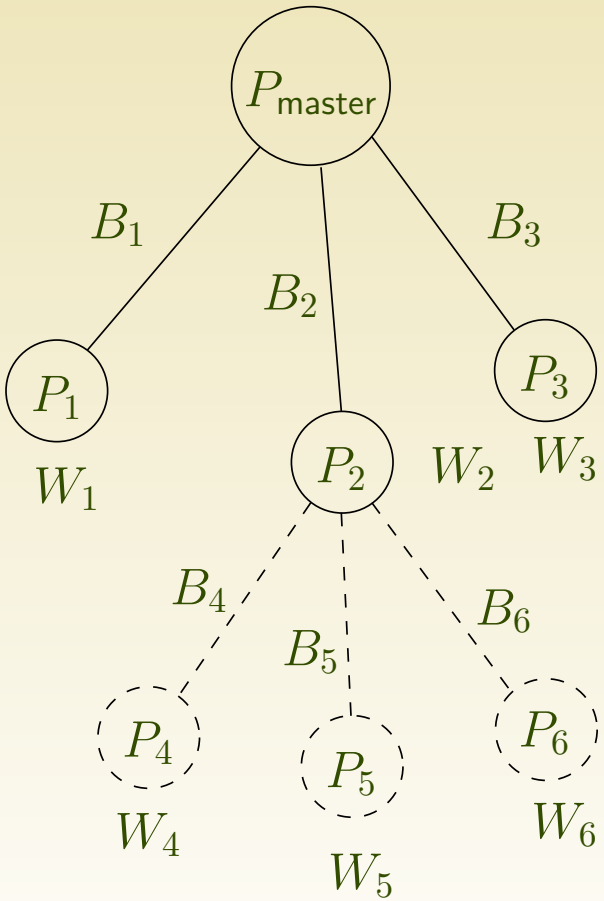
- Target platform: master-worker
  - Master P_{master} holds all tasks initially
  - Workers: $P_1, \ldots, P_p$
  - Father of P_u: $P_{f(u)}$
- Bandwidth of $P_{f(u)} \rightarrow P_u$: B_u (in MB/s)
- Speed of P_u: W_u (in MFlops/s)
- Communications and computations can be overlapped
- Multi-port communication model

Platform Model



- Target platform: master-worker
  - Master P_{master} holds all tasks initially
  - Workers: $P_1, \ldots, P_p$
  - Father of P_u: $P_{f(u)}$
- Bandwidth of $P_{f(u)} \rightarrow P_u$: B_u (in MB/s)
- Speed of P_u: W_u (in MFlops/s)
- Communications and computations can be overlapped
- Multi-port communication model

Platform Model



- Target platform: master-worker
  - Master P_{master} holds all tasks initially
  - Workers: $P_1, \ldots, P_p$
  - Father of P_u: $P_{f(u)}$
- Bandwidth of $P_{f(u)} \rightarrow P_u$: B_u (in MB/s)
- Speed of P_u: W_u (in MFlops/s)
- Communications and computations can be overlapped
- Multi-port communication model

Steady-state constraints

- Constraint for computations by P_i:

$$\forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i$$

- Number of bytes sent on $P_{f(i)} \rightarrow P_i$:

$$\forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M_{\rightarrow j}}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i$$

- Feasability constraints:

$$\forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0$$

We would like to maximize *all* throughputs D_k.

Defining fairness enables to go from a multi-criteria problem to a more classical mono-criteria problem.

Steady-state constraints

- Constraint for computations by P_i:

$$\forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i$$

- Number of bytes sent on $P_{f(i)} \rightarrow P_i$:

$$\forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M \rightarrow j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i$$

- Feasability constraints:

$$\forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0$$

We would like to maximize *all* throughputs D_k.

Defining fairness enables to go from a multi-criteria problem to a more classical mono-criteria problem.

Steady-state constraints

- Constraint for computations by P_i:

$$\forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i$$

- Number of bytes sent on $P_{f(i)} \rightarrow P_i$:

$$\forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M \rightarrow j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i$$

- Feasability constraints:

$$\forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0$$

We would like to maximize *all* throughputs D_k.

Defining fairness enables to go from a multi-criteria problem to a more classical mono-criteria problem.

Steady-state constraints

- Constraint for computations by P_i:

$$\forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i$$

- Number of bytes sent on $P_{f(i)} \rightarrow P_i$:

$$\forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M \rightarrow j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i$$

- Feasability constraints:

$$\forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0$$

We would like to maximize *all* throughputs D_k.

Defining fairness enables to go from a multi-criteria problem to a more classical mono-criteria problem.

Steady-state constraints

- Constraint for computations by P_i:

$$\forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i$$

- Number of bytes sent on $P_{f(i)} \rightarrow P_i$:

$$\forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M \rightarrow j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i$$

- Feasability constraints:

$$\forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0$$

We would like to maximize *all* throughputs D_k.

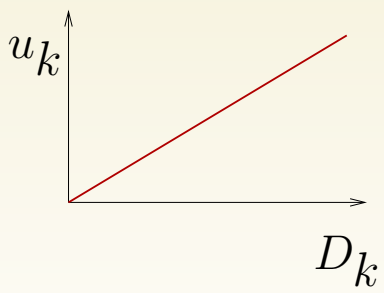
Defining fairness enables to go from a multi-criteria problem to a more classical mono-criteria problem.

Utility Function

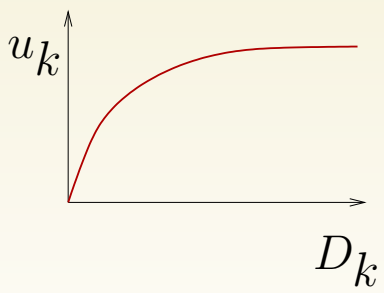
In a general context, each application is characterized by a utility function u_k defined on $(\alpha_n^{(k)})_{1 \leqslant k \leqslant K, 1 \leqslant n \leqslant N}$.
In our context, the utility is simply defined by

$$u_k(\alpha) = \sum_n \alpha_n^{(k)} = D_k$$

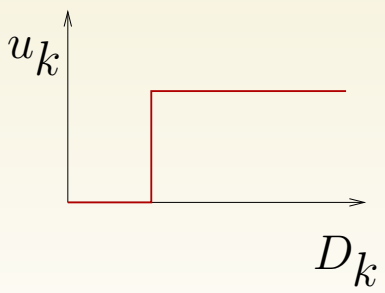
But we could perfectly imagine other utility functions:



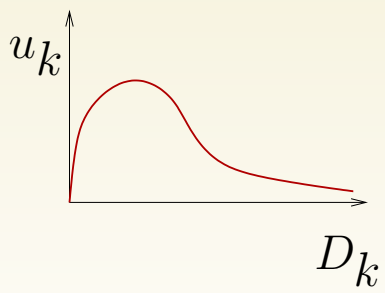
linear



Voice over IP



threshold



price-accounting

Multi-player Optimality

Our goal is to find scheduling strategies such that the utility of each user is maximized.

Definition: Pareto-optimality.

A vector of strategy is said to be Pareto optimal if it is impossible to strictly increase the utility of a player without strictly decreasing the one of another. In other words, $\widetilde{\alpha}$ is Pareto optimal iff:

$$\forall \alpha, \exists i, u_i(\alpha) > u_i(\widetilde{\alpha}) \Rightarrow \exists j, u_j(\alpha) < u_j(\widetilde{\alpha})$$

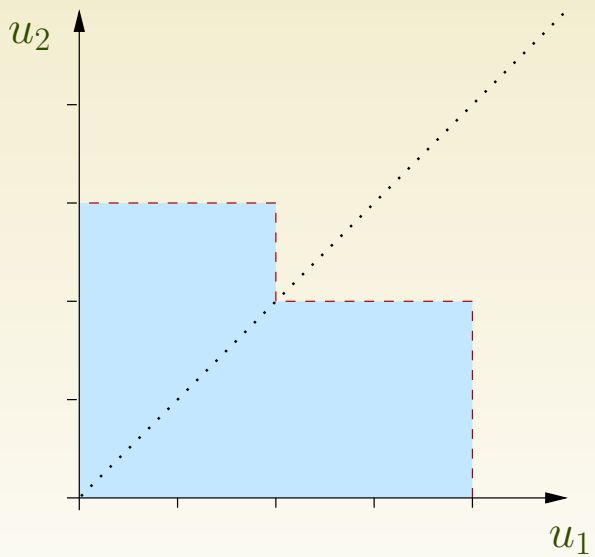
Multi-player Optimality

Our goal is to find scheduling strategies such that the utility of each user is maximized.

Definition: Pareto-optimality.

A vector of strategy is said to be Pareto optimal if it is impossible to strictly increase the utility of a player without strictly decreasing the one of another. In other words, $\widetilde{\alpha}$ is Pareto optimal iff:

$$\forall \alpha, \exists i, u_i(\alpha) > u_i(\widetilde{\alpha}) \Rightarrow \exists j, u_j(\alpha) < u_j(\widetilde{\alpha})$$



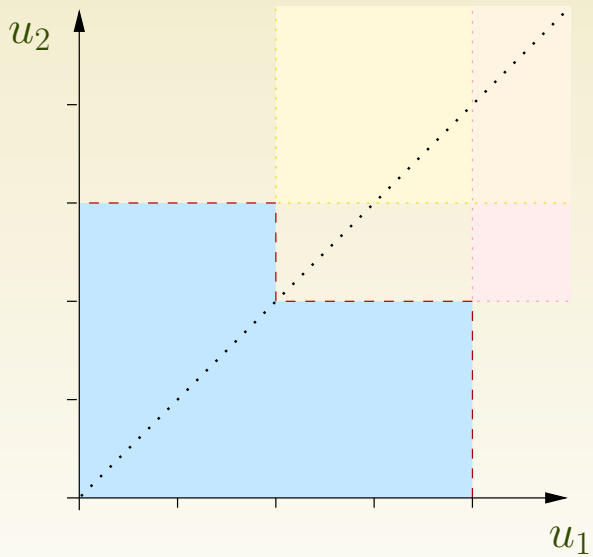
Multi-player Optimality

Our goal is to find scheduling strategies such that the utility of each user is maximized.

Definition: Pareto-optimality.

A vector of strategy is said to be Pareto optimal if it is impossible to strictly increase the utility of a player without strictly decreasing the one of another. In other words, $\widetilde{\alpha}$ is Pareto optimal iff:

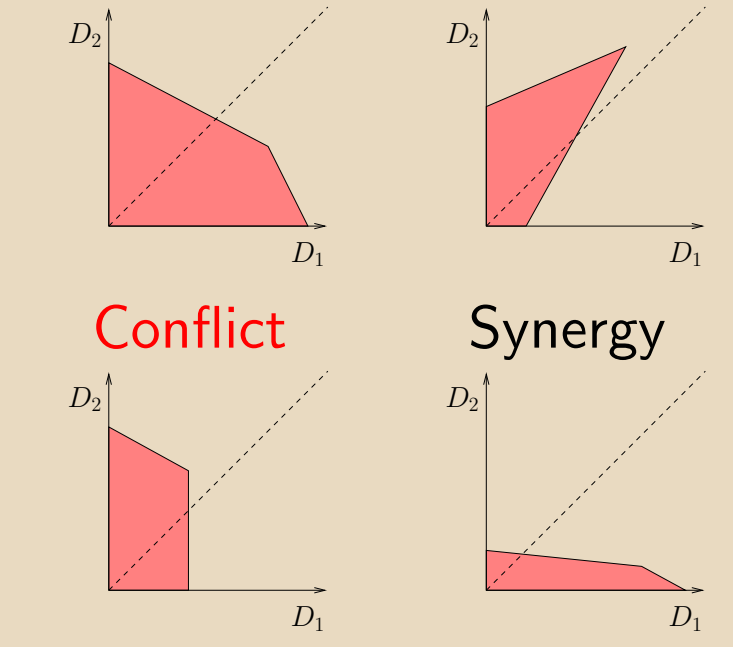
$$\forall \alpha, \exists i, u_i(\alpha) > u_i(\widetilde{\alpha}) \Rightarrow \exists j, u_j(\alpha) < u_j(\widetilde{\alpha})$$



Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

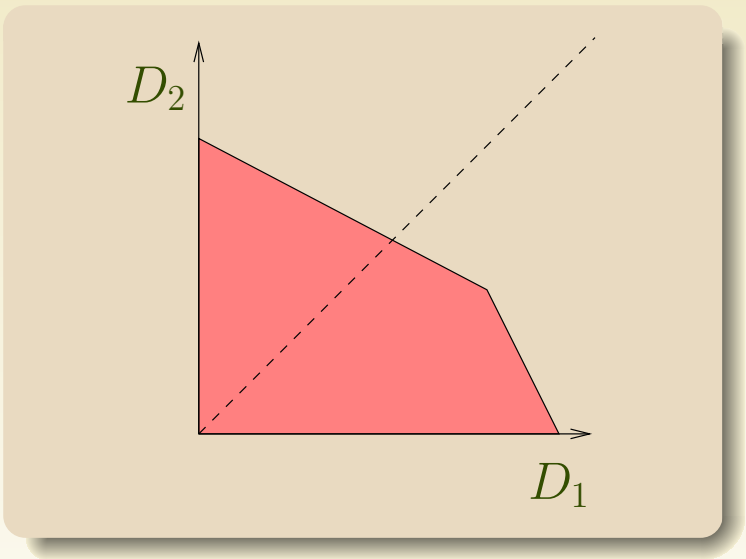
$$\begin{aligned}
\alpha^{(1)} \cdot c^{(1)} + \alpha^{(2)} \cdot c^{(2)} &\leqslant W_u \\
\alpha^{(1)} \cdot b^{(1)} + \alpha^{(2)} \cdot b^{(2)} &\leqslant B_u \\
\alpha^{(1)} &\geqslant 0 \\
\alpha^{(2)} &\geqslant 0
\end{aligned}$$

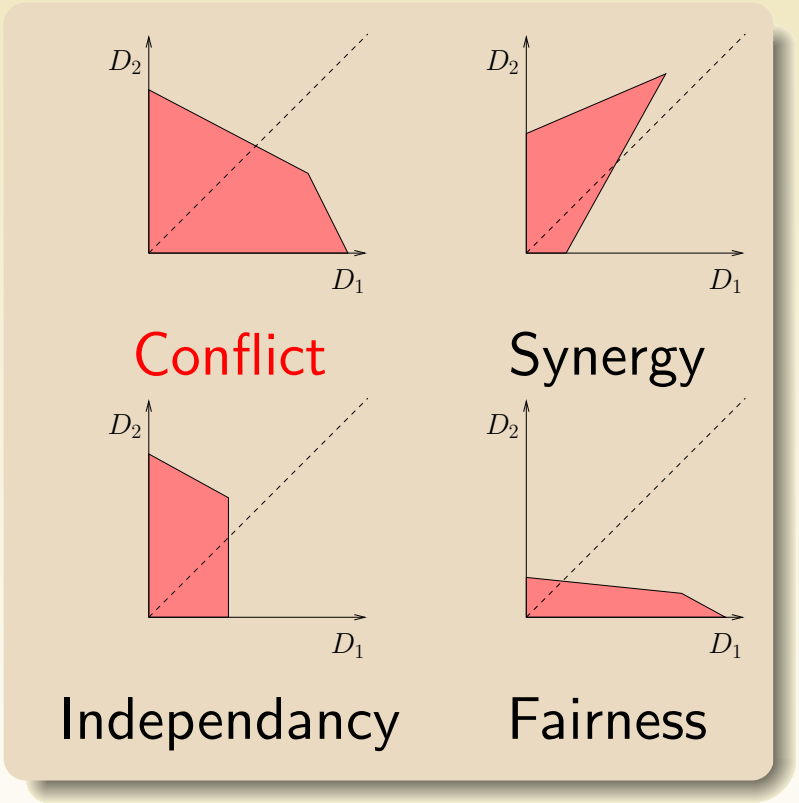


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

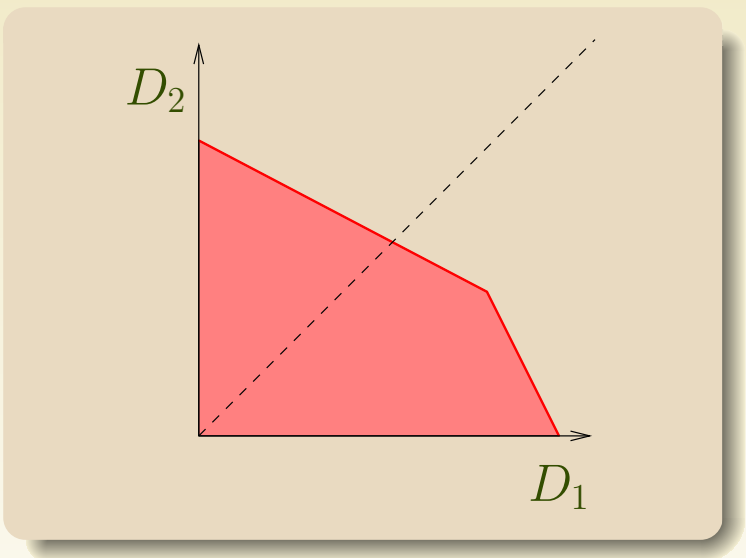


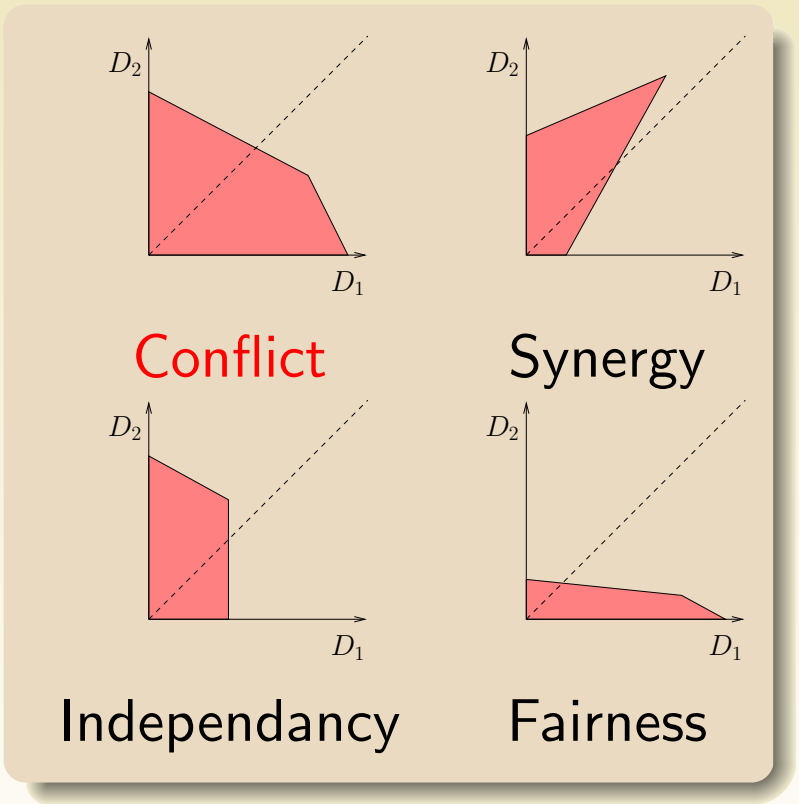


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

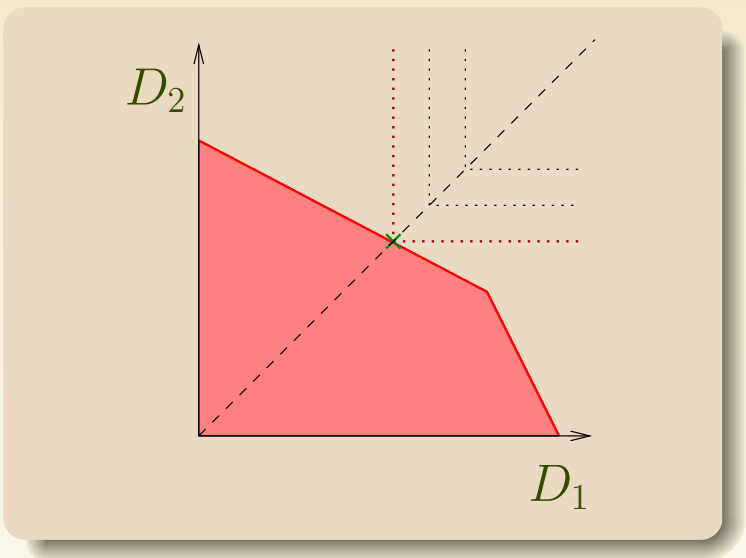


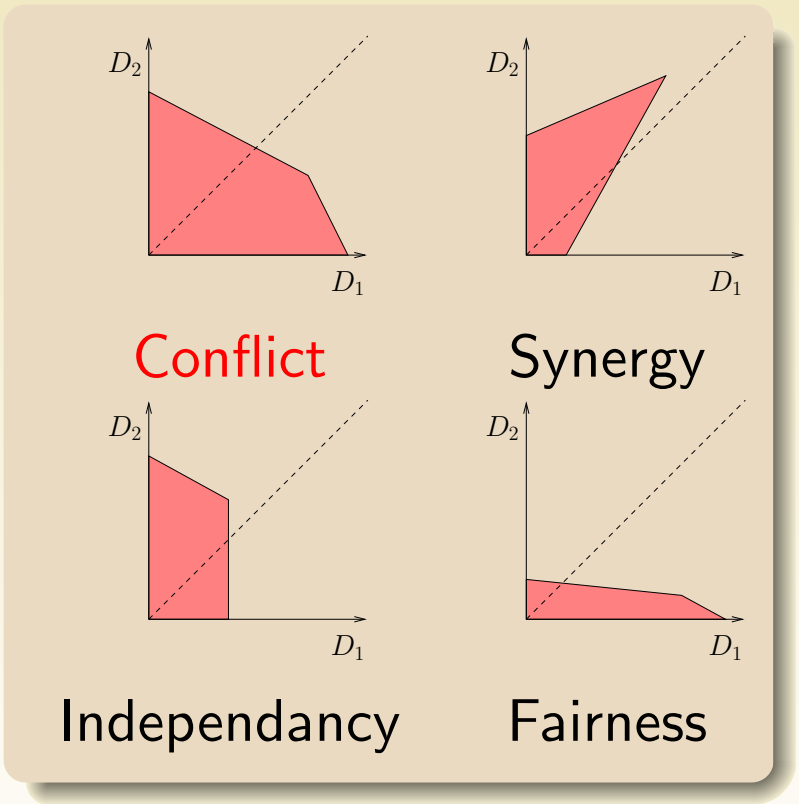


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

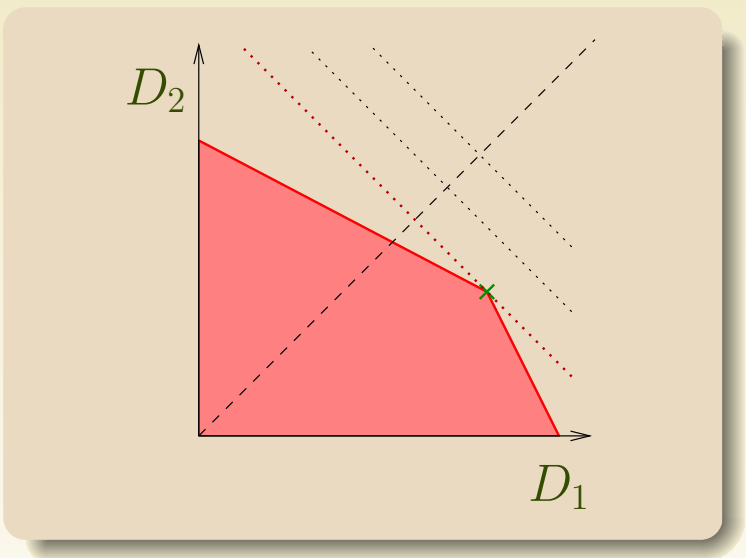


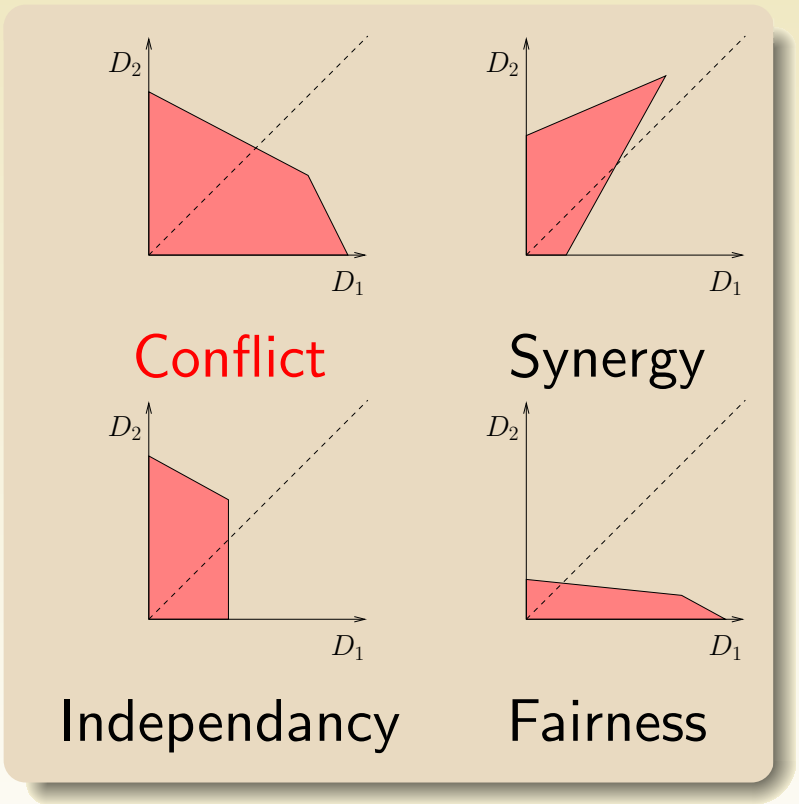


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

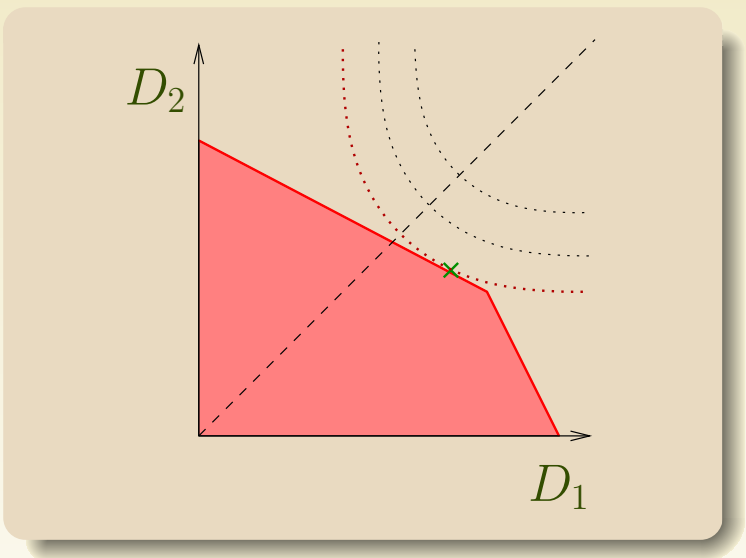


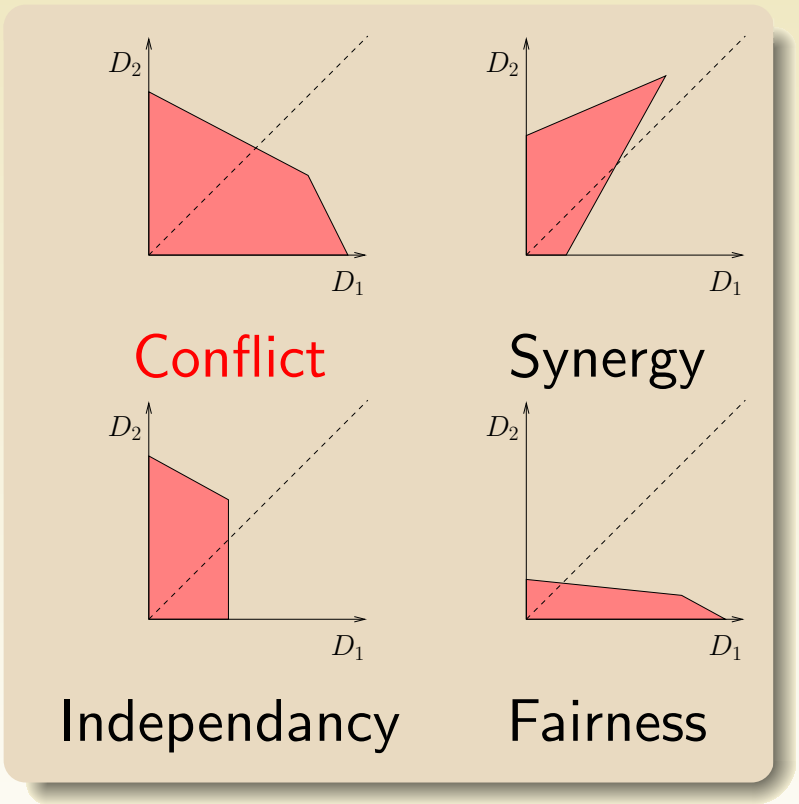


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

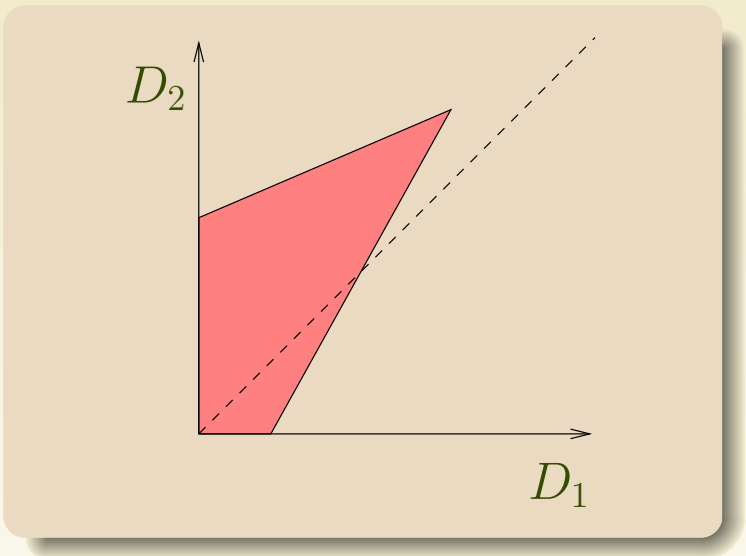


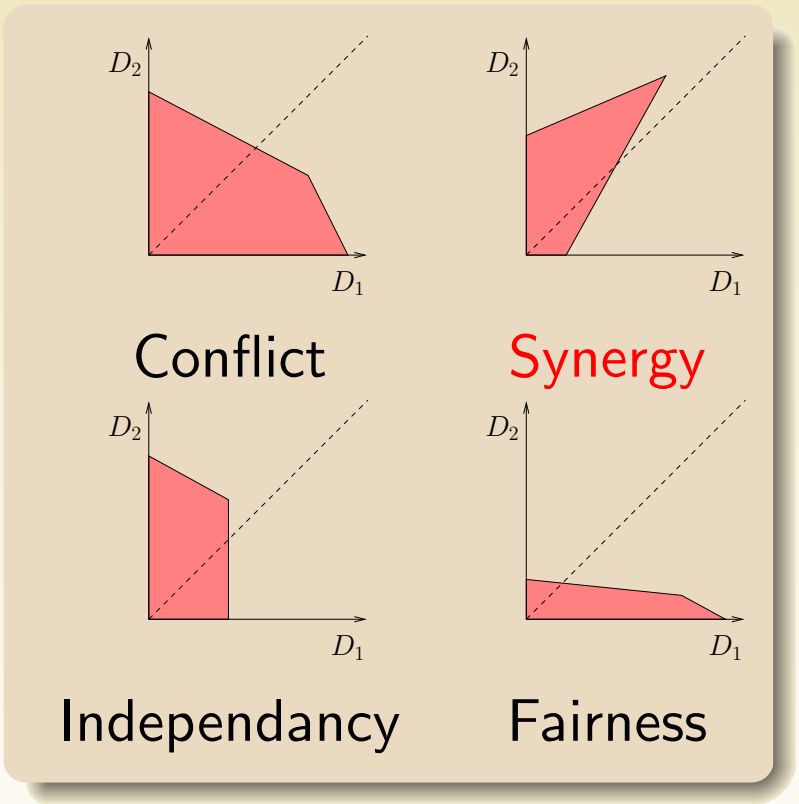


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

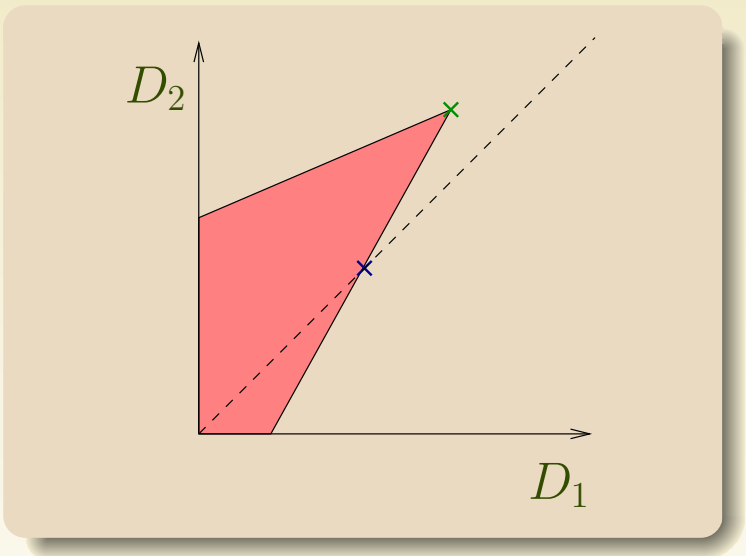


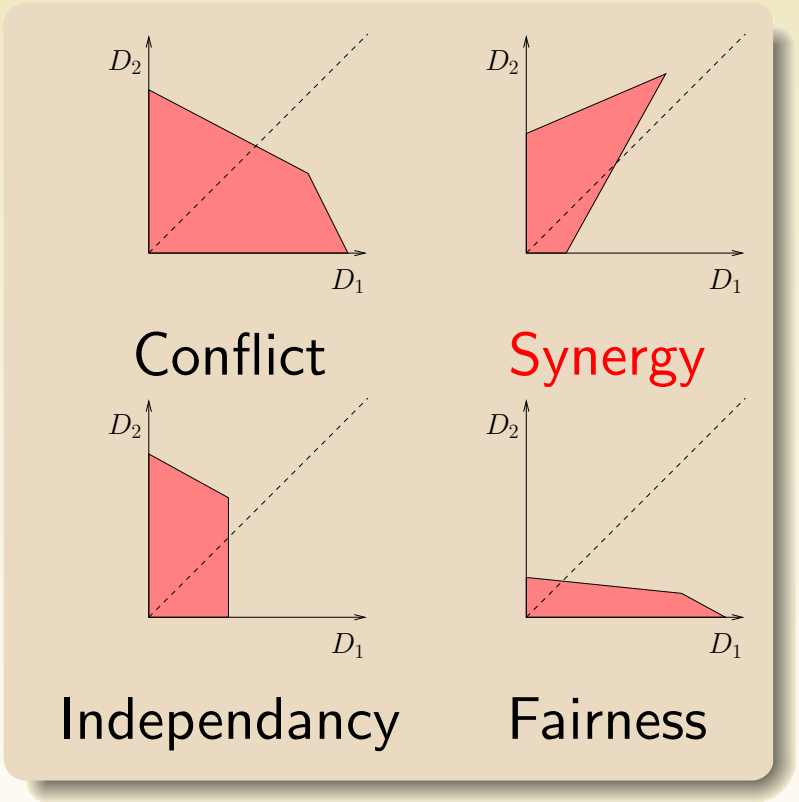


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

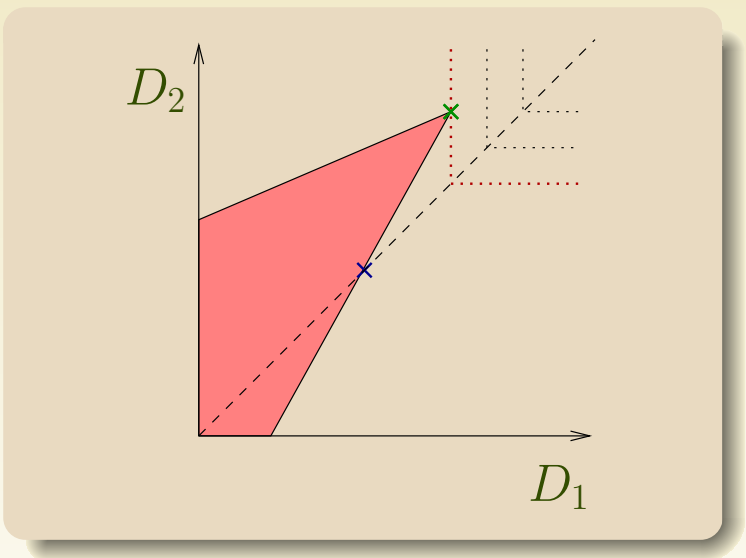


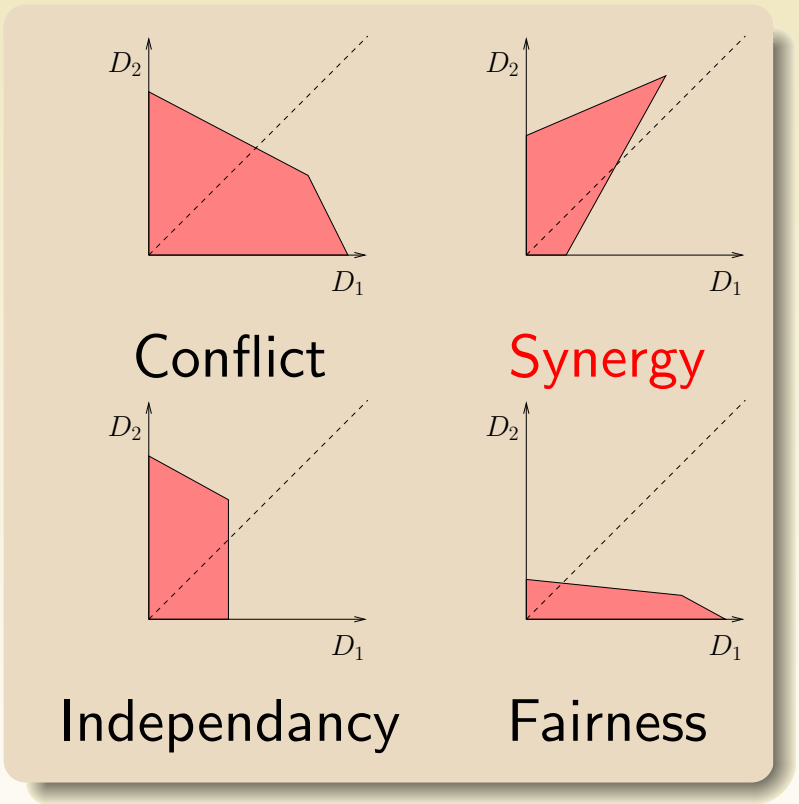


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

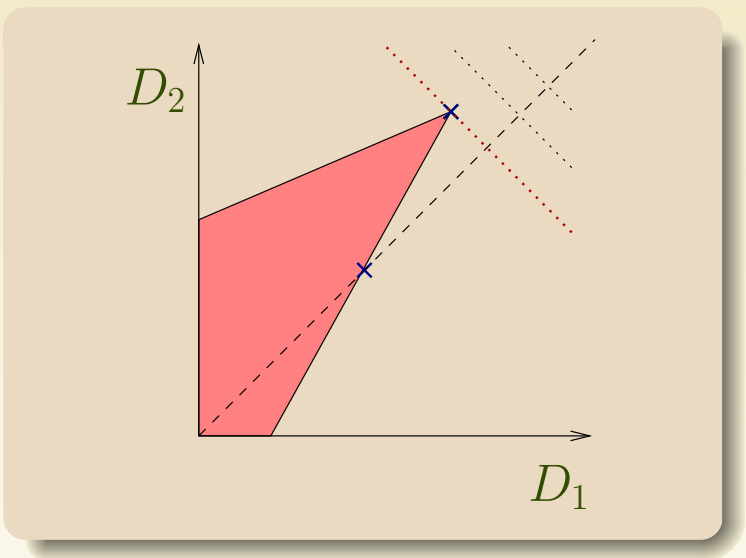


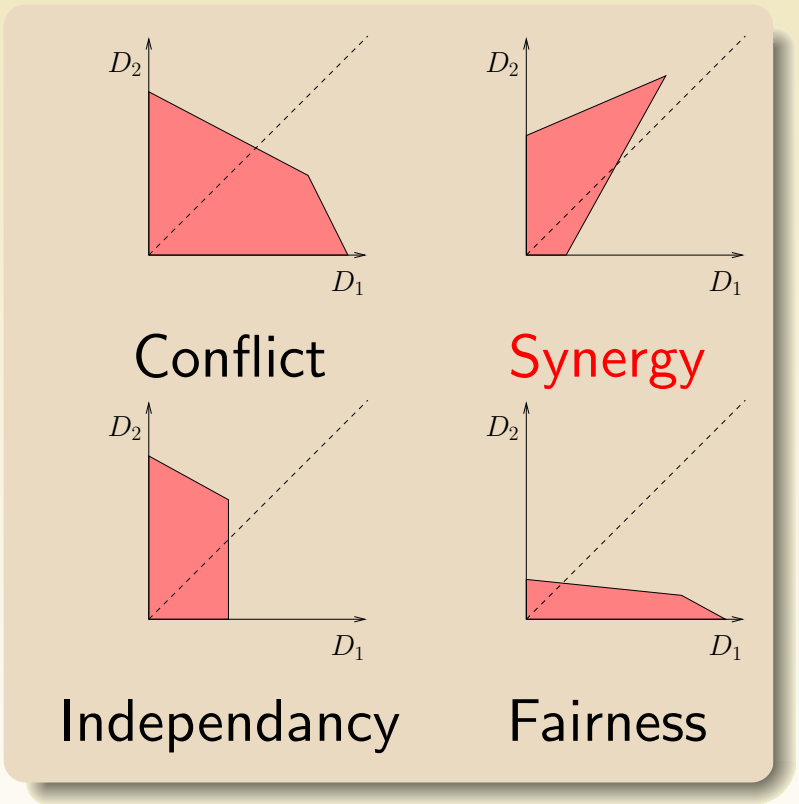


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

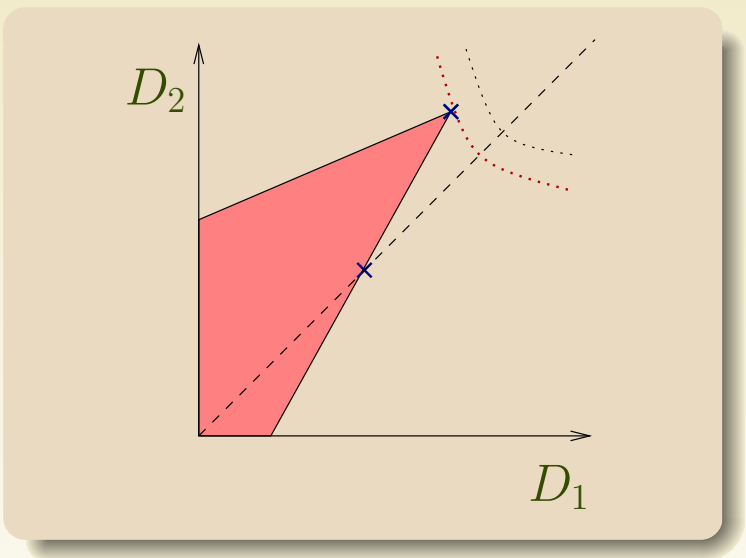


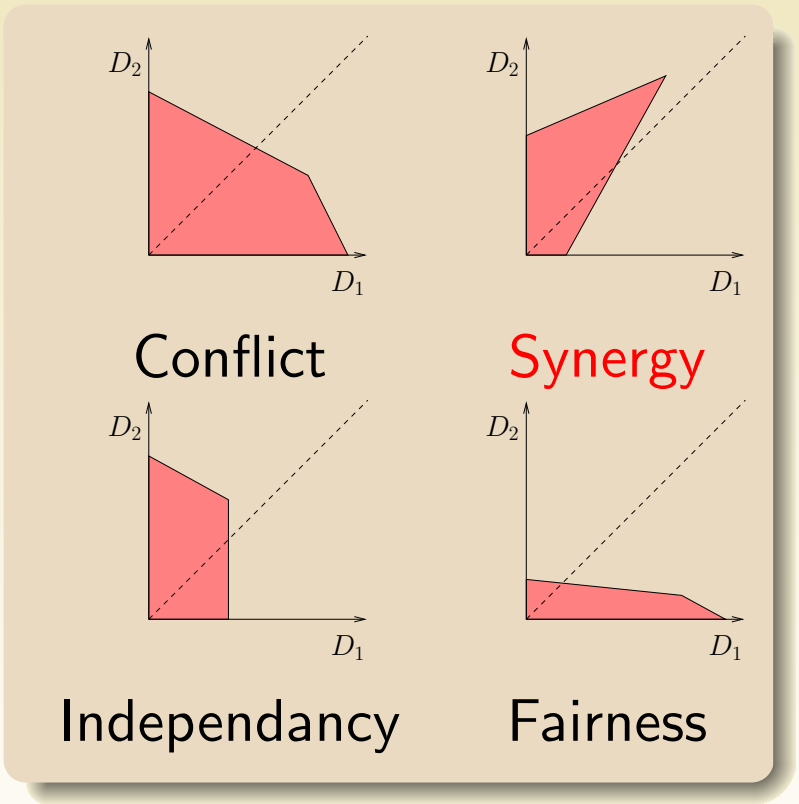


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

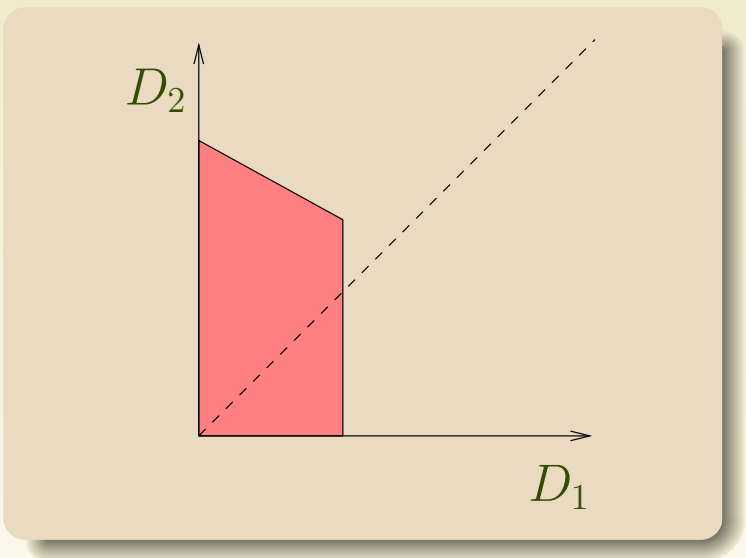


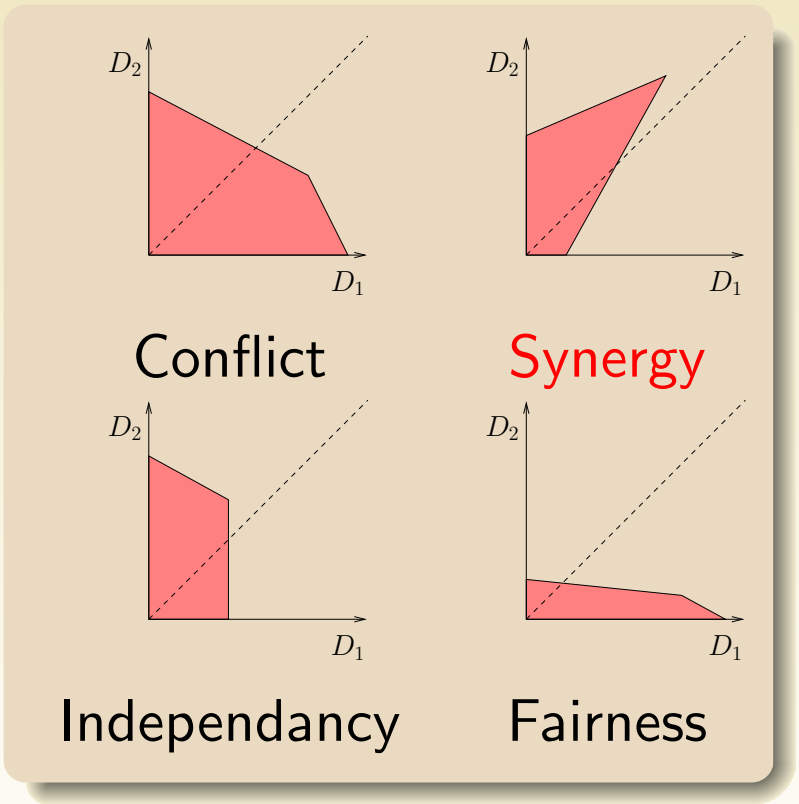


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

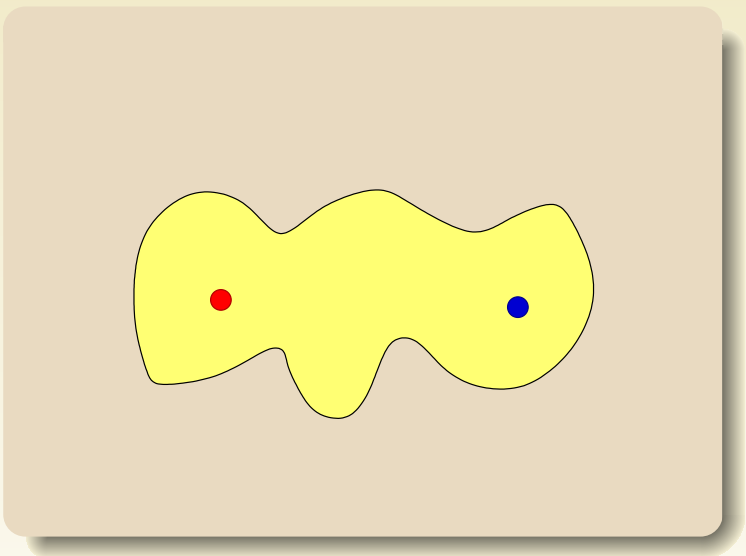


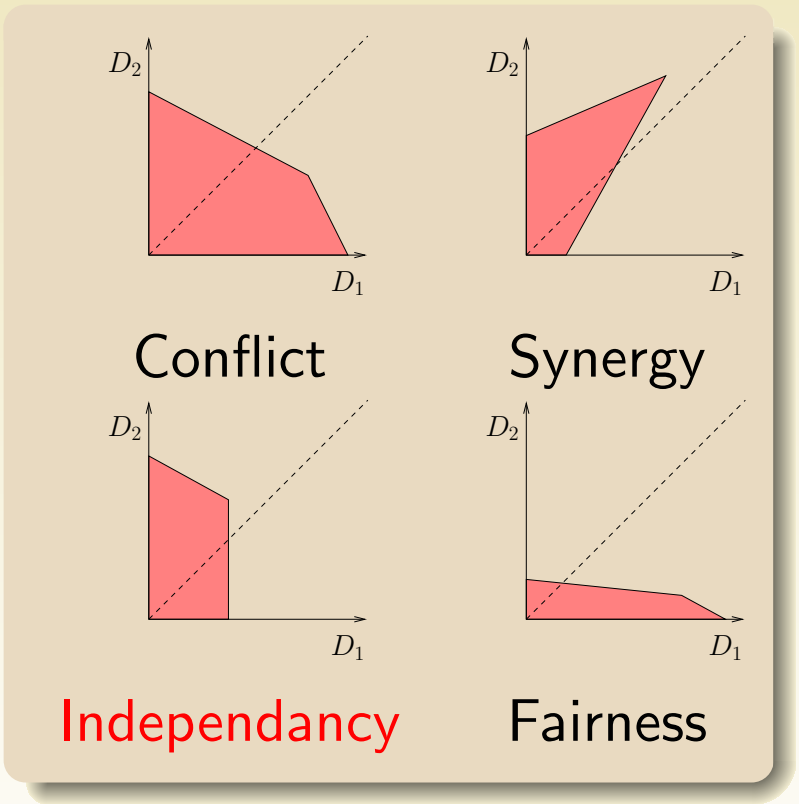


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

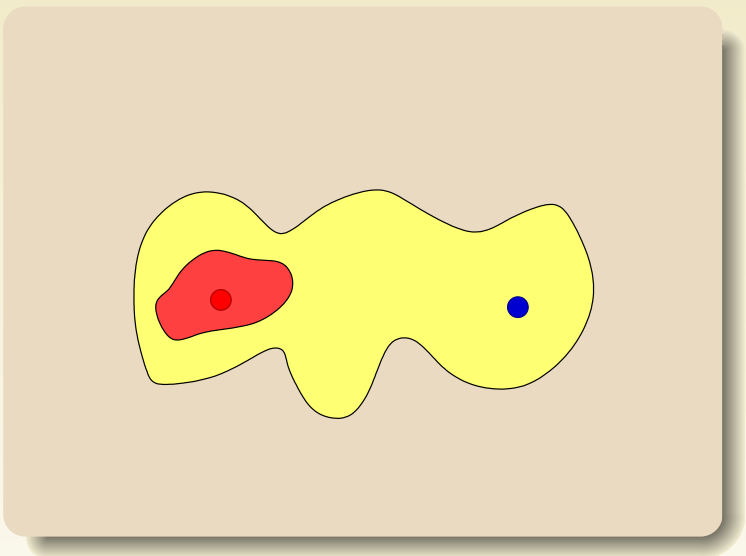


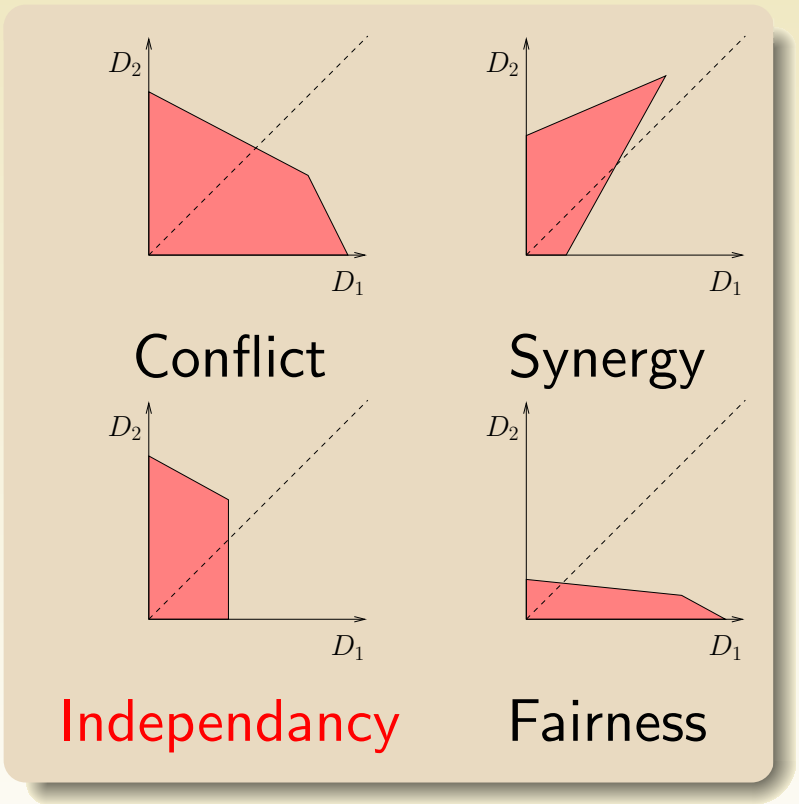


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

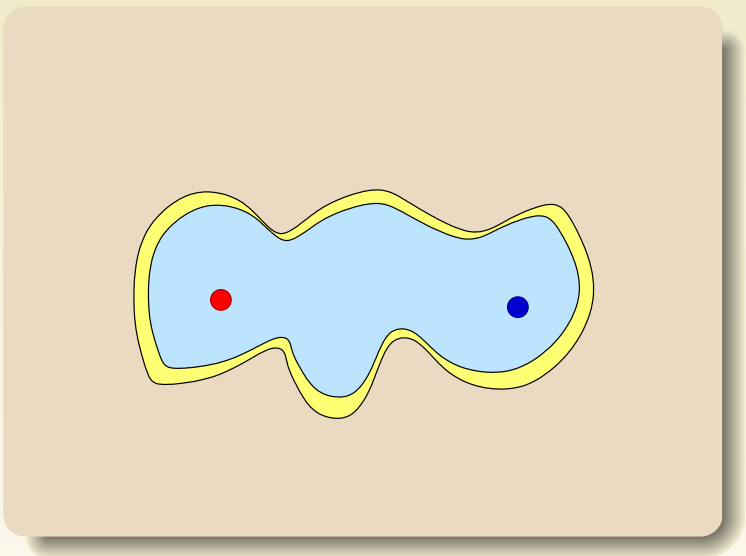


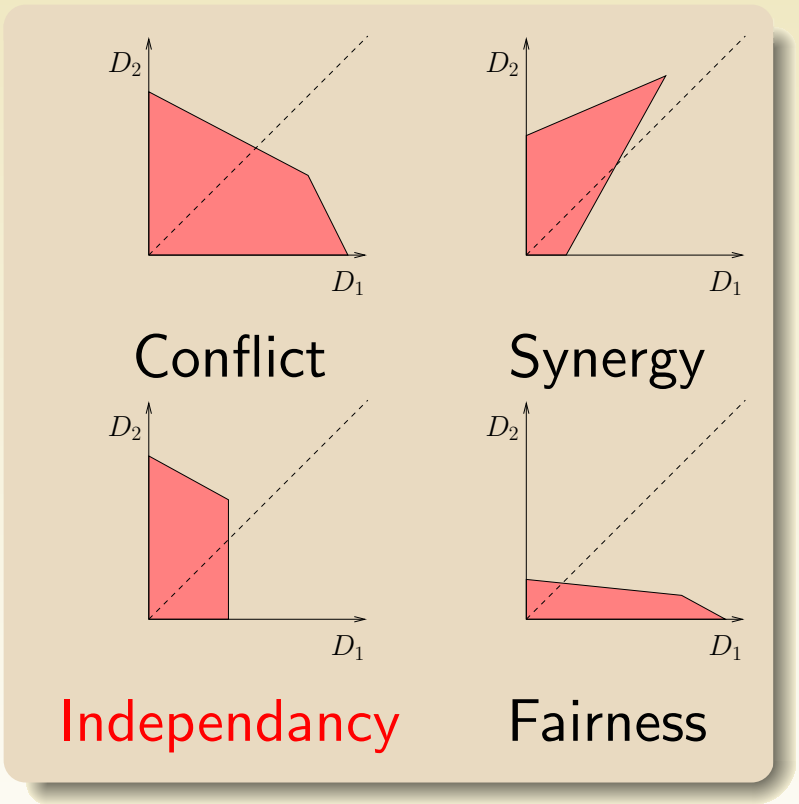


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

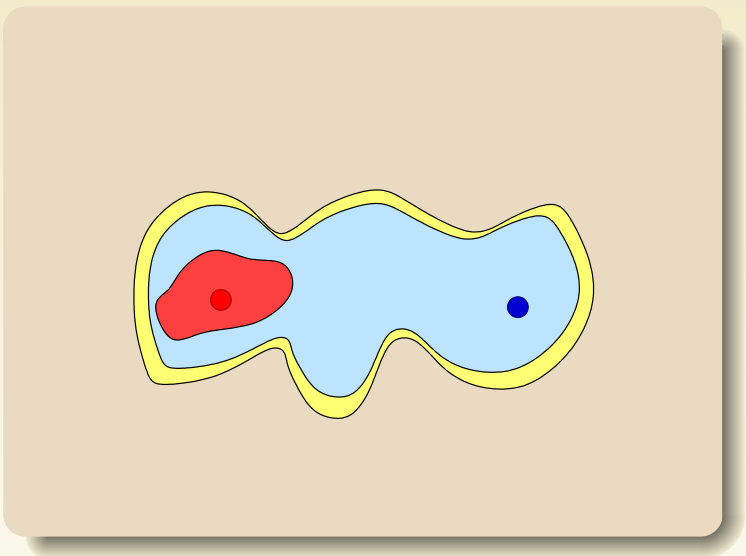


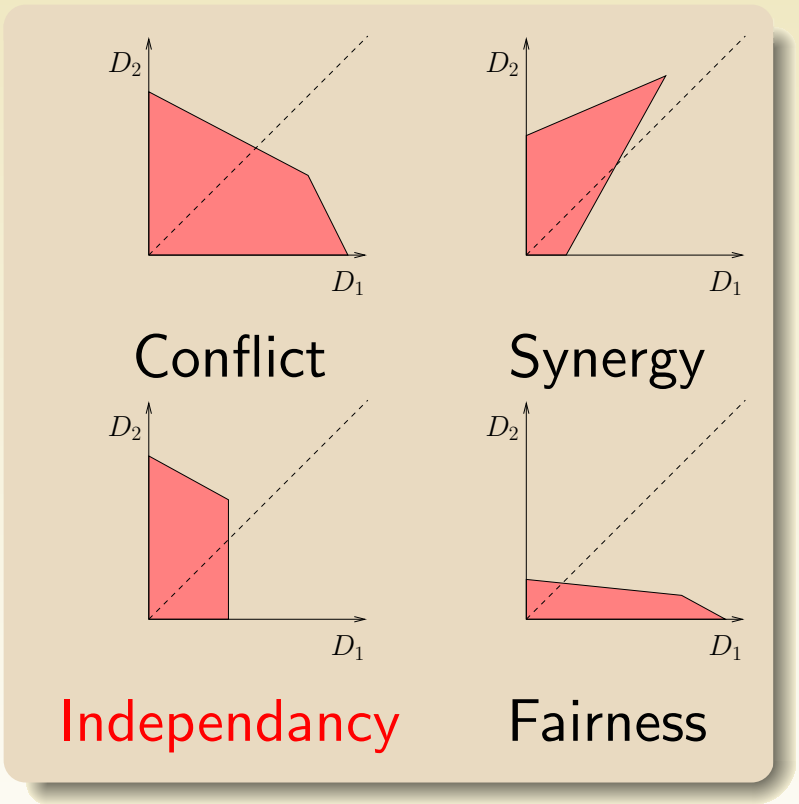


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

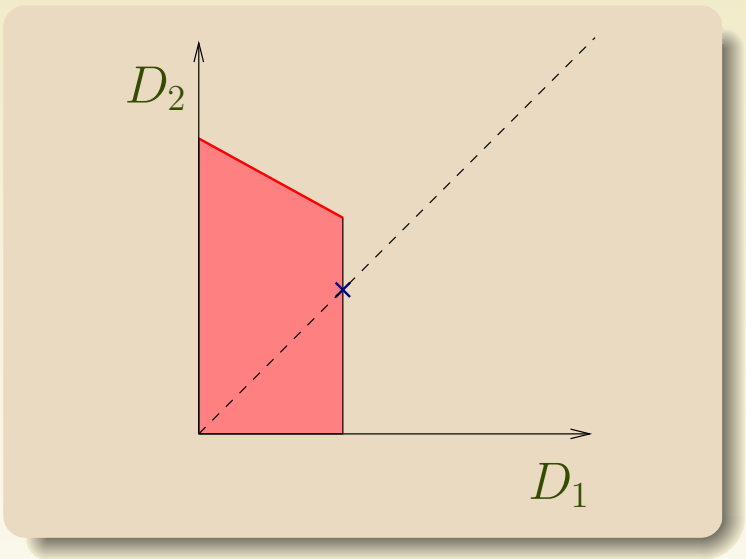


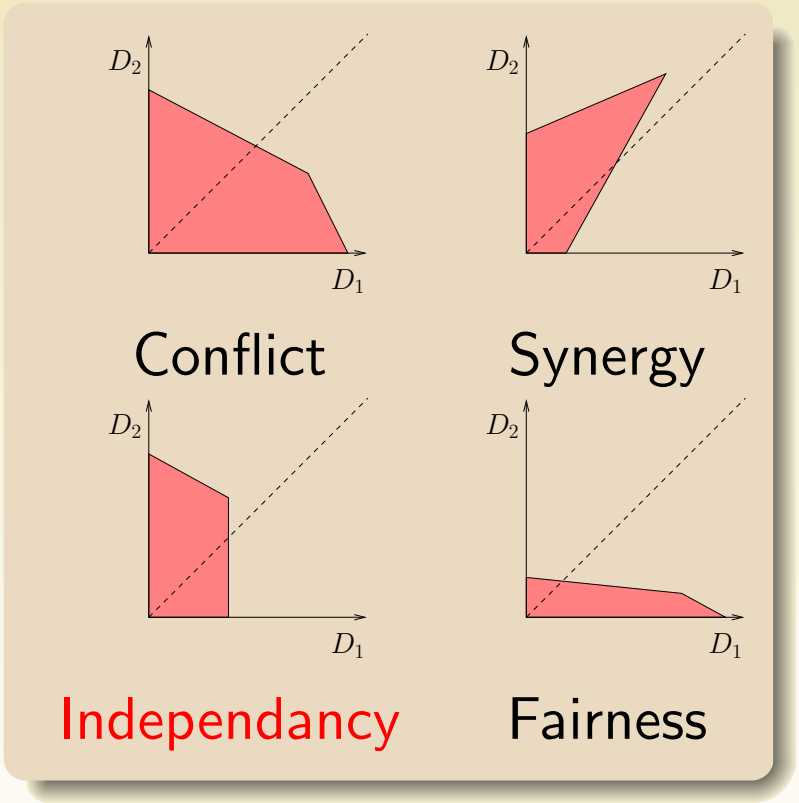


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

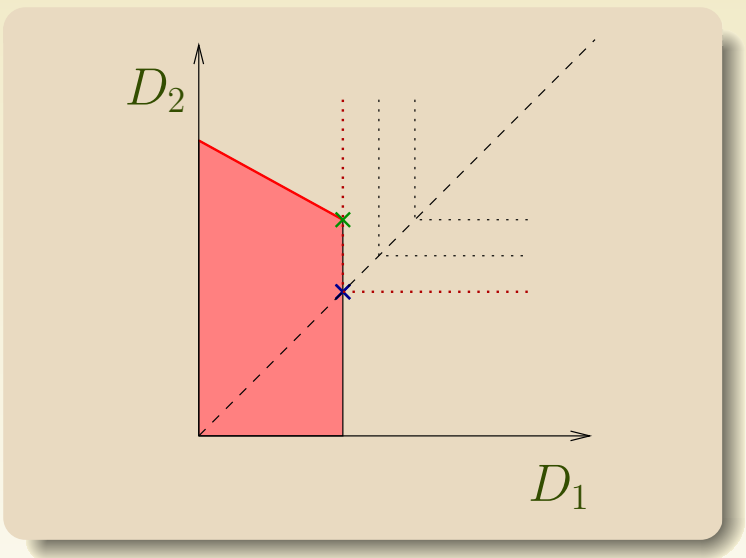


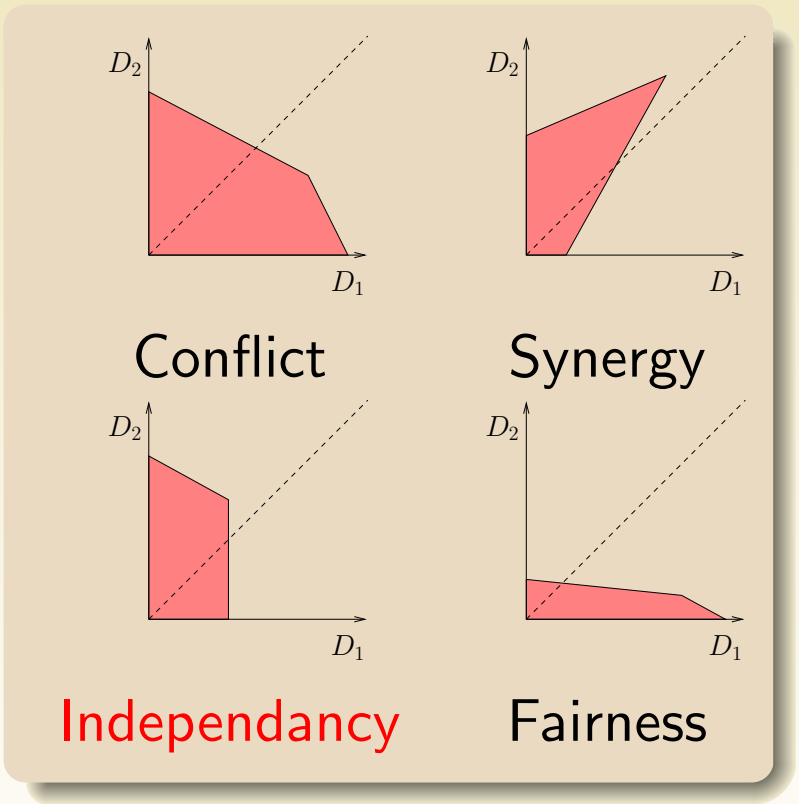


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

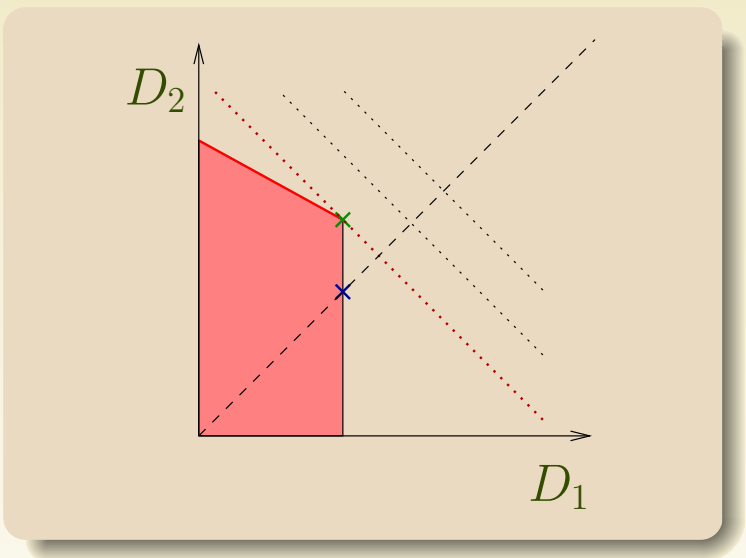


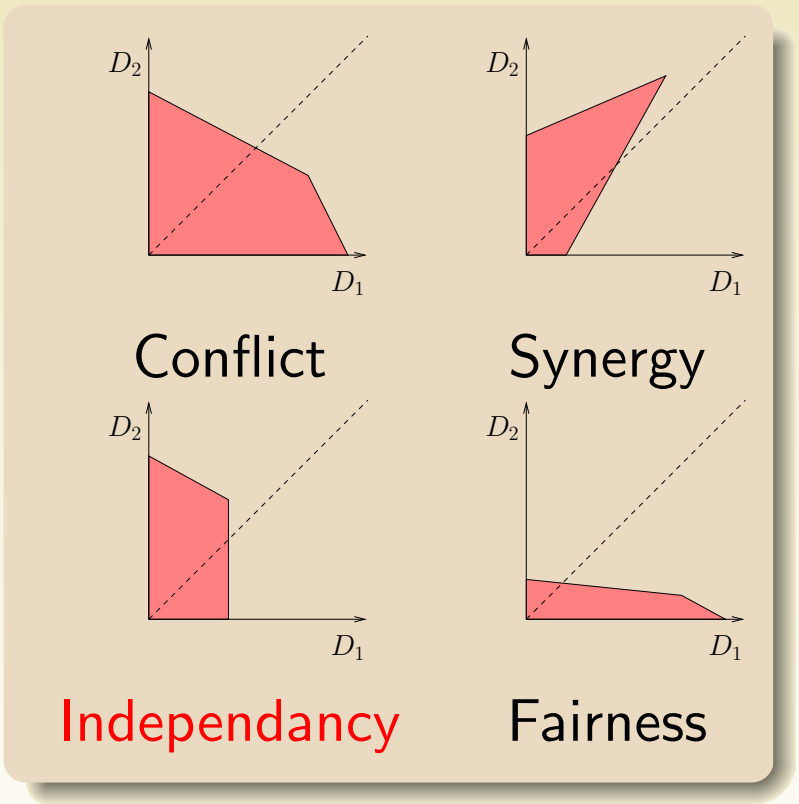


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

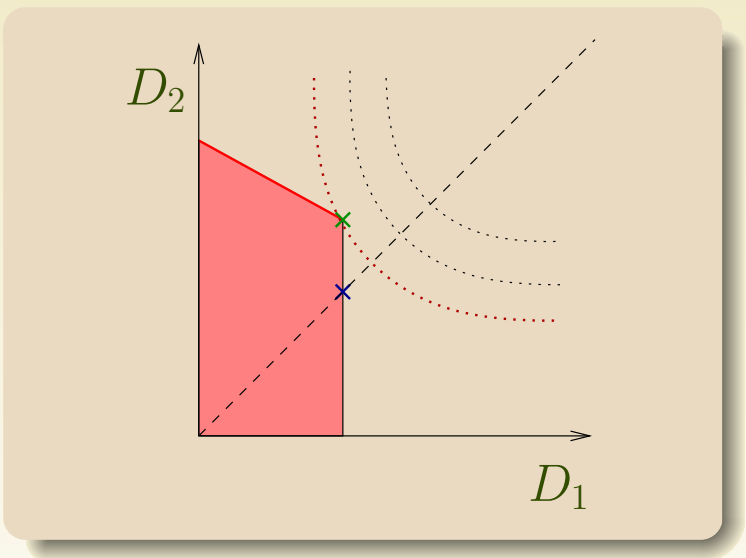


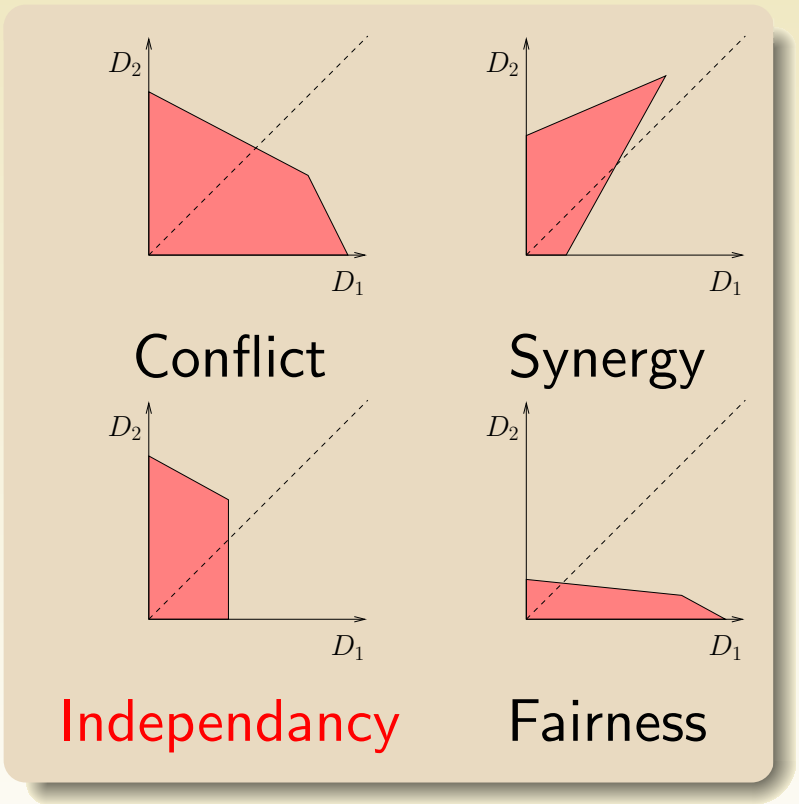


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

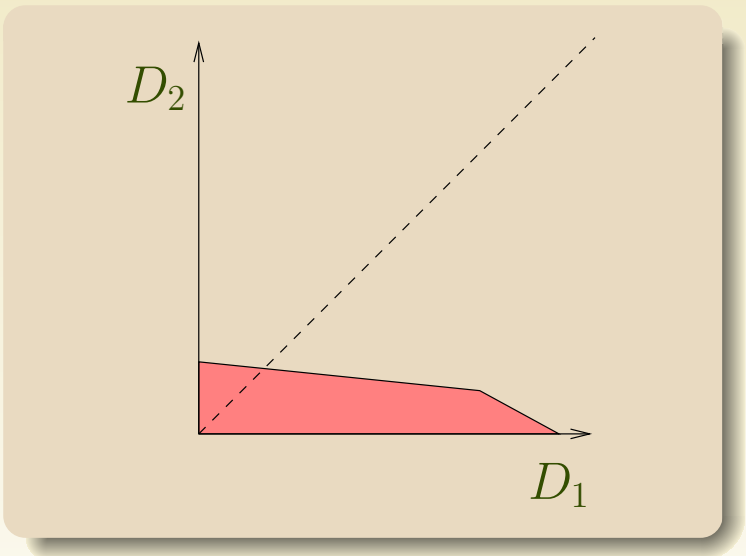


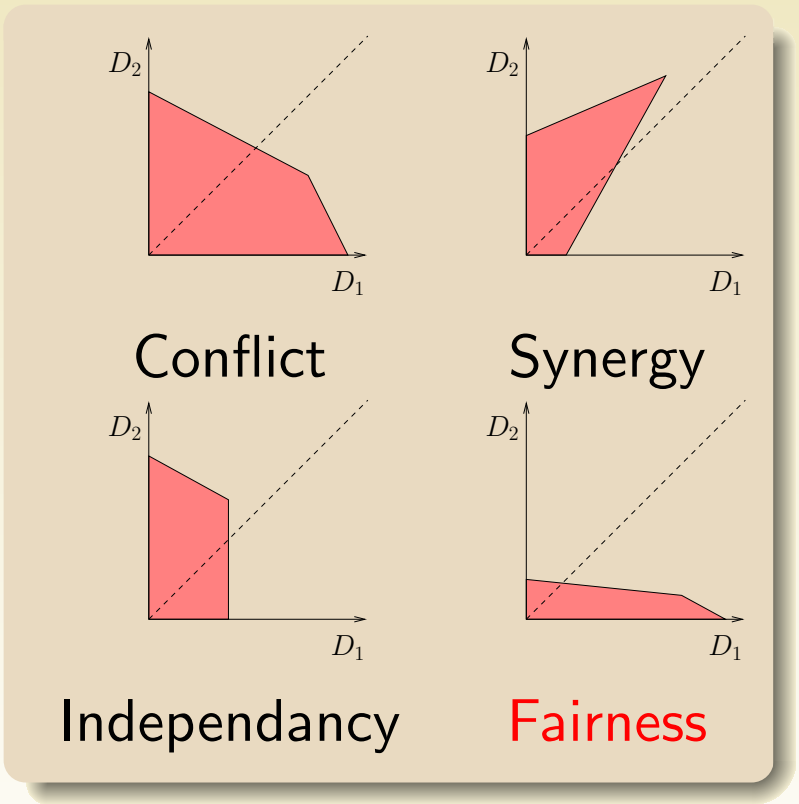


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

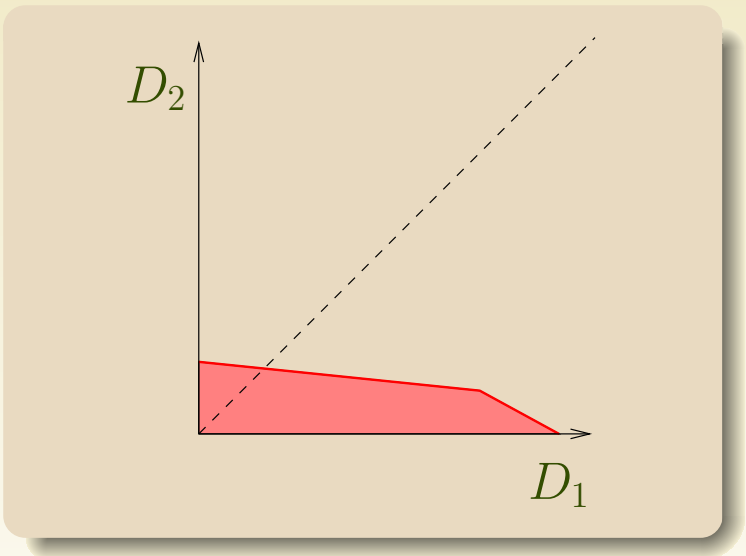


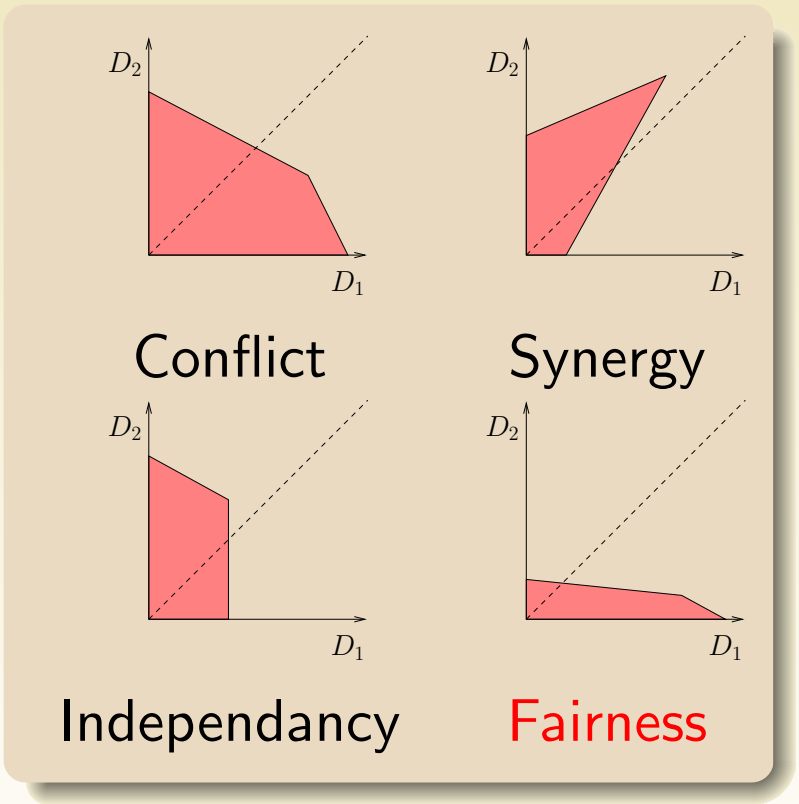


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

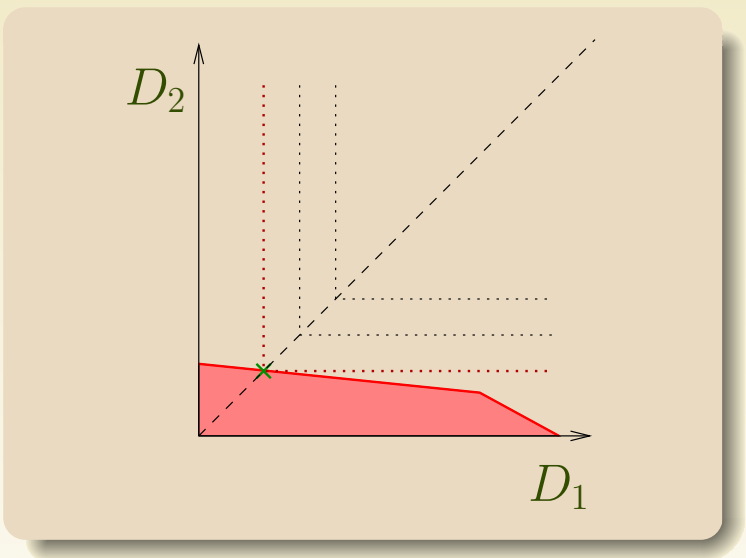


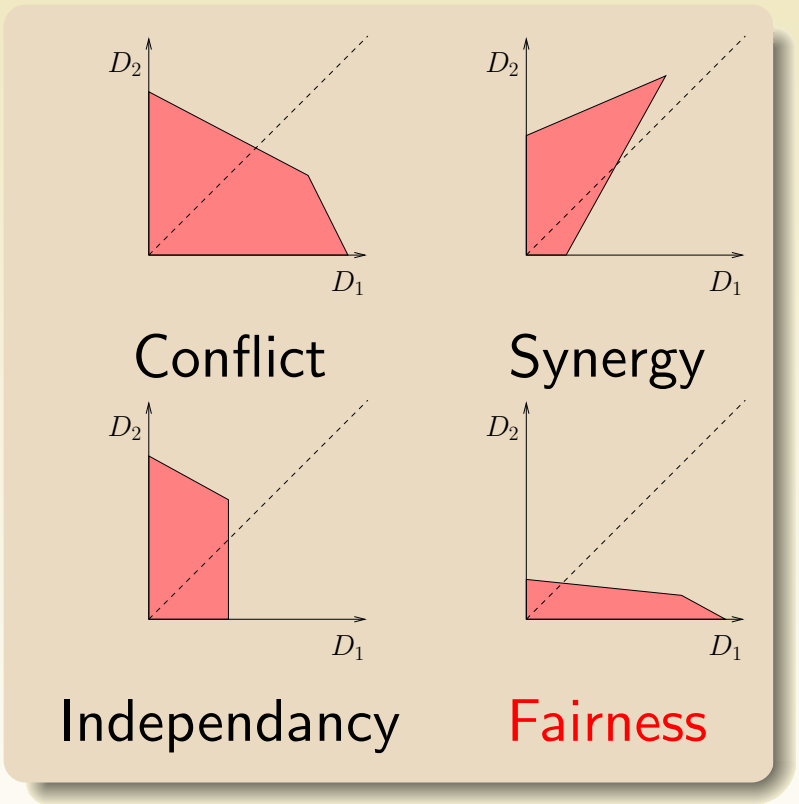


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

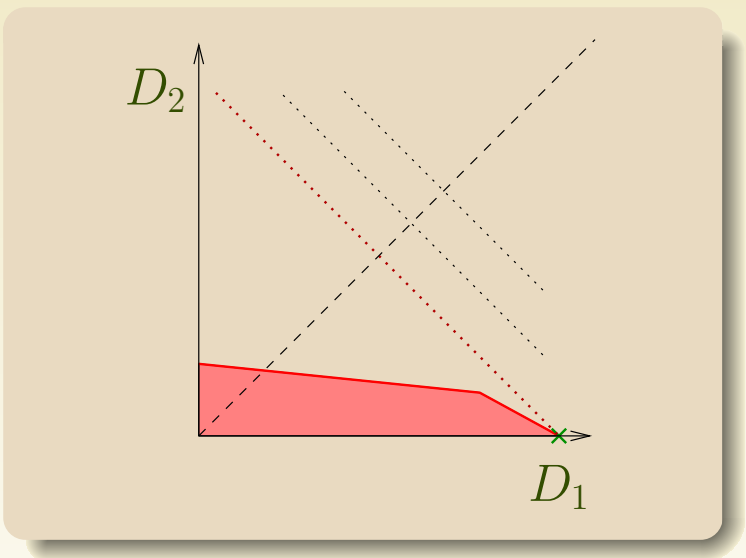


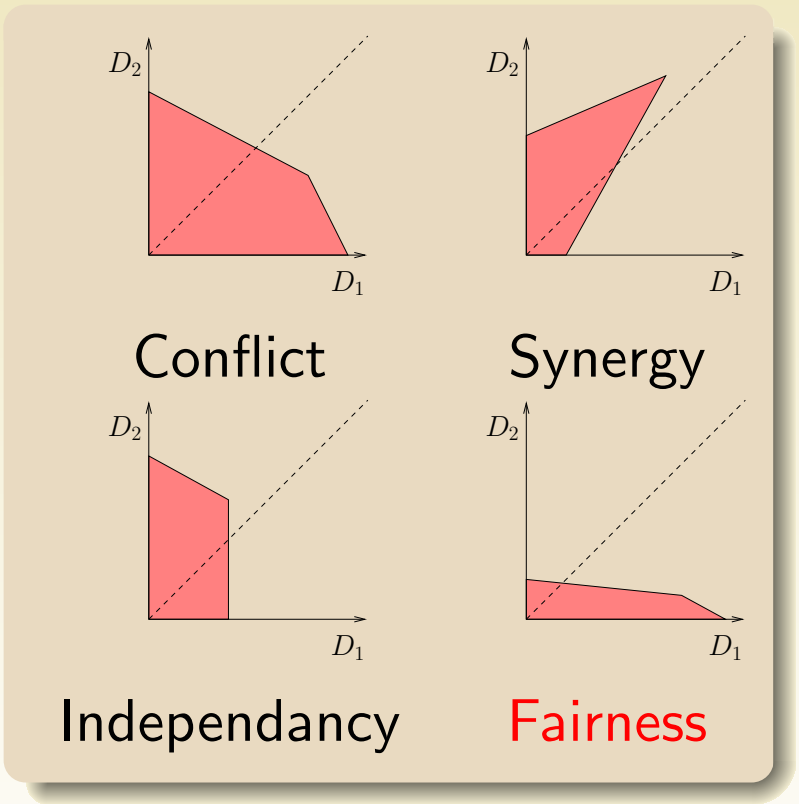


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

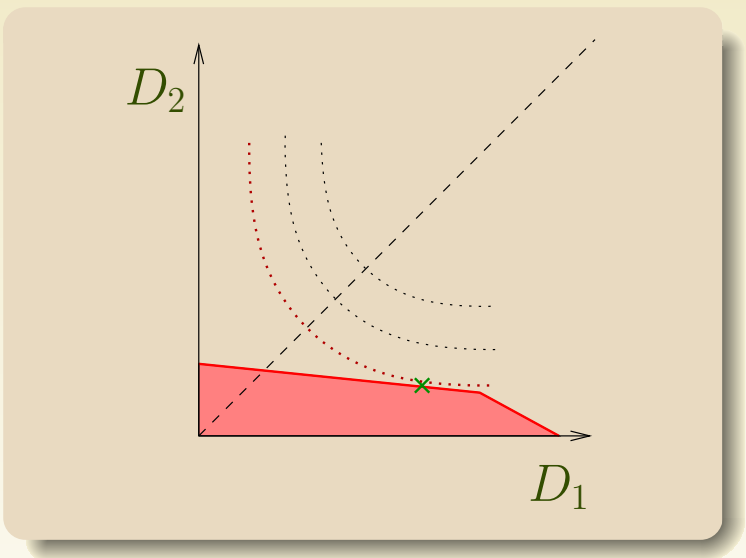


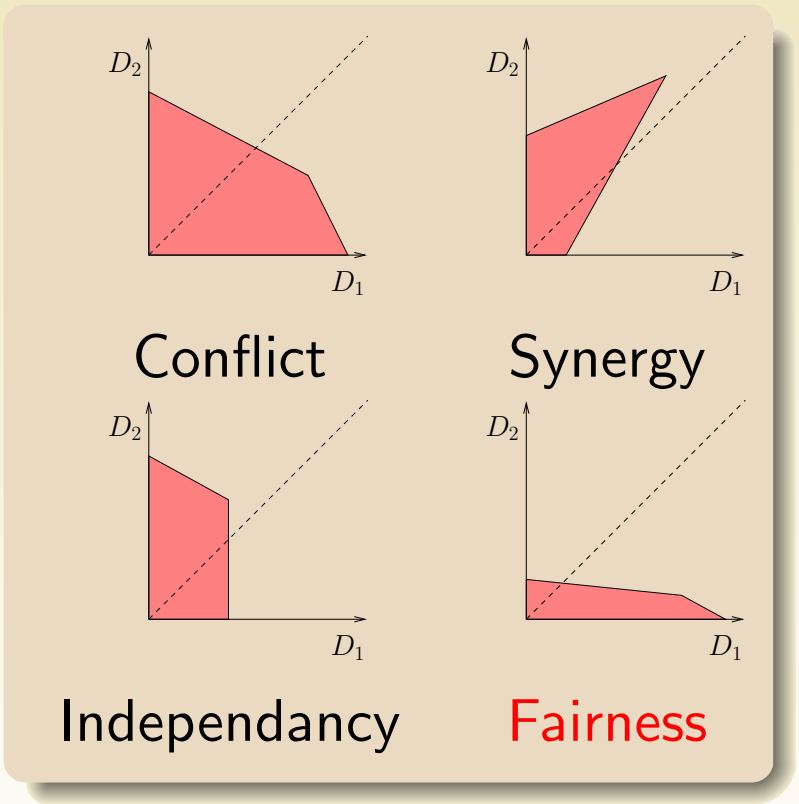


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

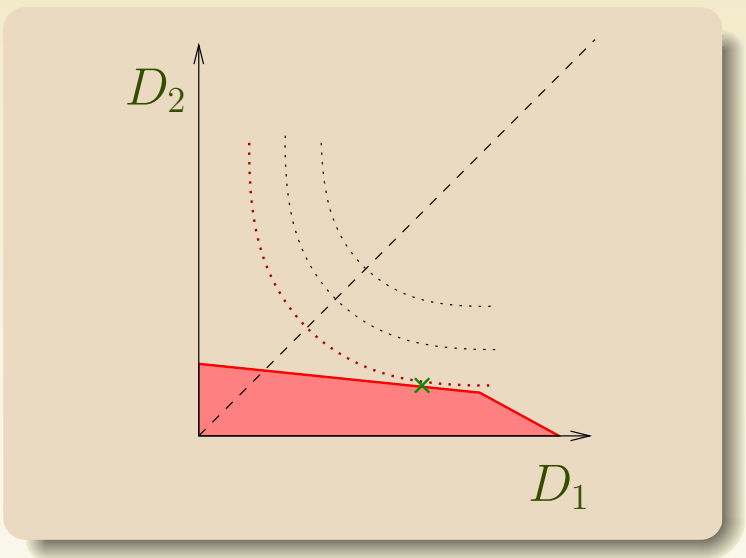


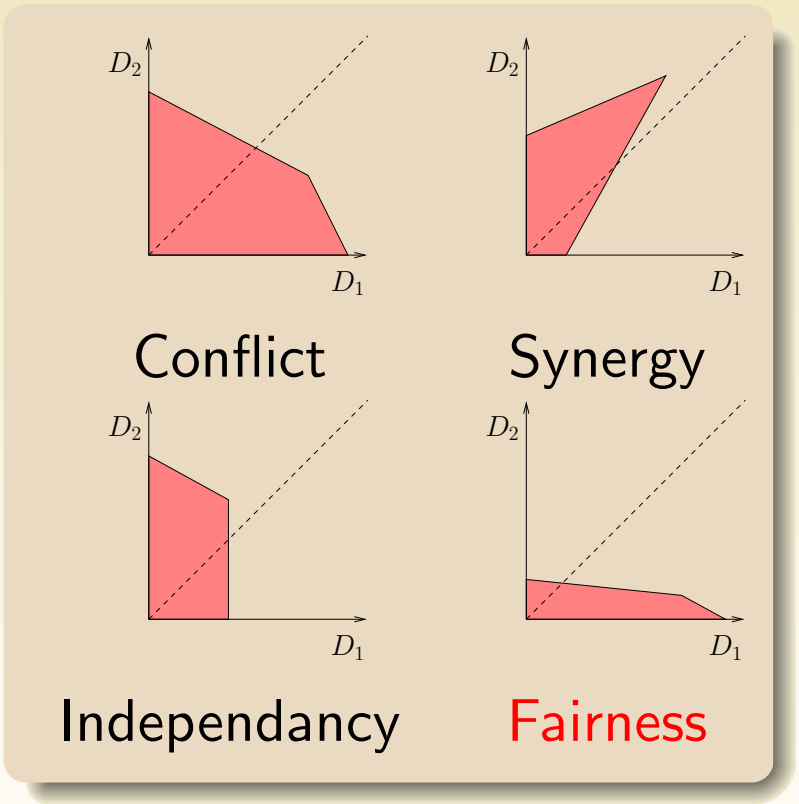


Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Utility Set and Fairness

How can fair be defined? Does it always mean *"give the same thing to everyone"*? How can efficiency be defined?

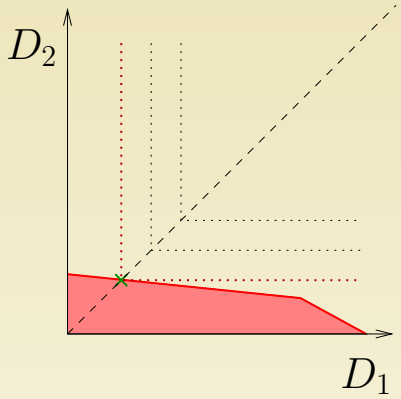




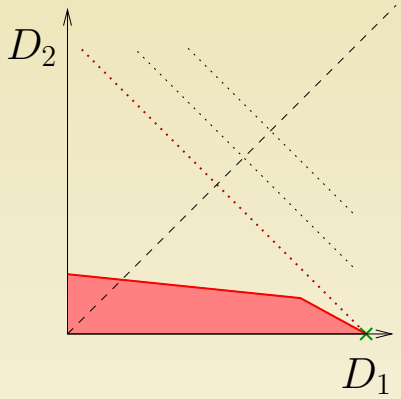
Fairness can be seen as the trade-off between individual satisfaction and global satisfaction.

Defining Fairness for our framework

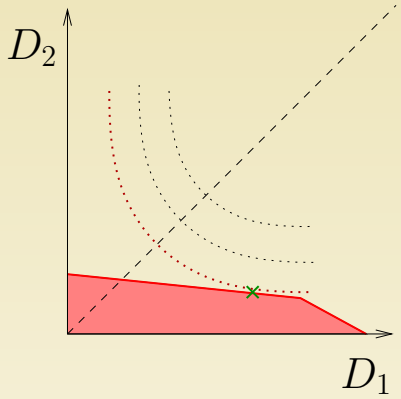
- Types d'équité:



Max min
$\min_k D_k$



Social welfare
$\sum_k D_k$



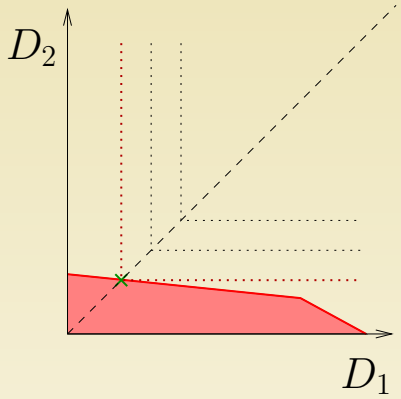
Proportional
$\prod_k D_k$

- A few problems with max-min fairness:
  - Not efficient when applications are very different
  - Seems to be hard to reach on such platforms [rr2005-45]
- Let's try proportional fairness!

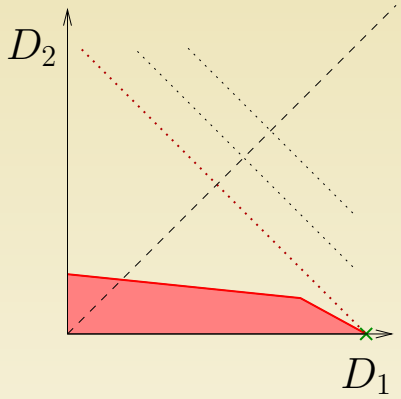
$$\text{MAXIMIZE} \left(\sum_{k \in A} \ln D_k \right)$$

Defining Fairness for our framework

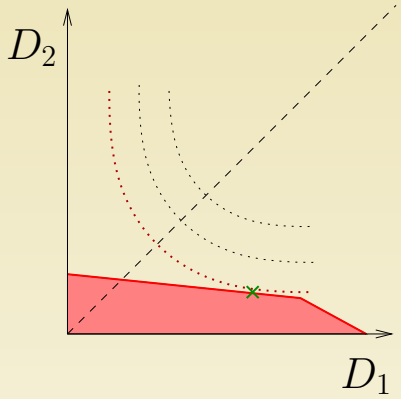
- Types d'équité:



Max min
$\min_k D_k$



Social welfare
$\sum_k D_k$



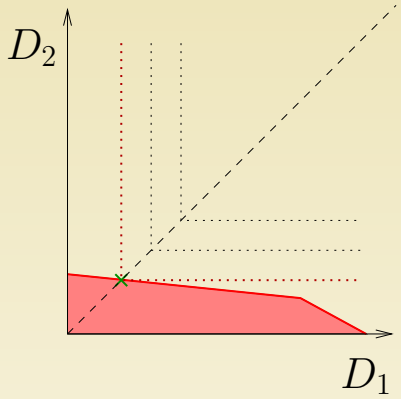
Proportional
$\prod_k D_k$

- A few problems with max-min fairness:
  - Not efficient when applications are very different
  - Seems to be hard to reach on such platforms [rr2005-45]
- Let's try proportional fairness!

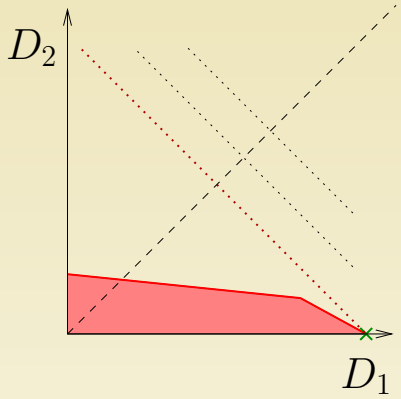
$$\text{MAXIMIZE} \left(\sum_{k \in A} \ln D_k \right)$$

Defining Fairness for our framework

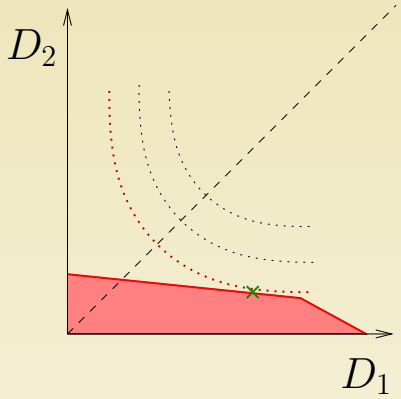
- Types d'équité:



Max min
$\min_k D_k$



Social welfare
$\sum_k D_k$



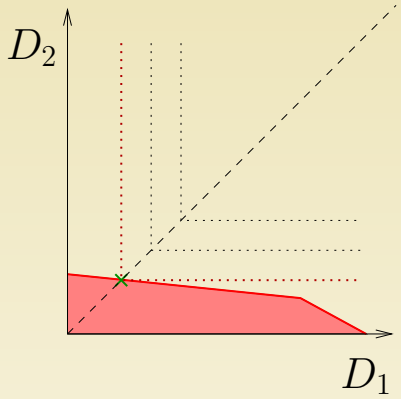
Proportional
$\prod_k D_k$

- A few problems with max-min fairness:
  - Not efficient when applications are very different
  - Seems to be hard to reach on such platforms [rr2005-45]
- Let's try proportional fairness!

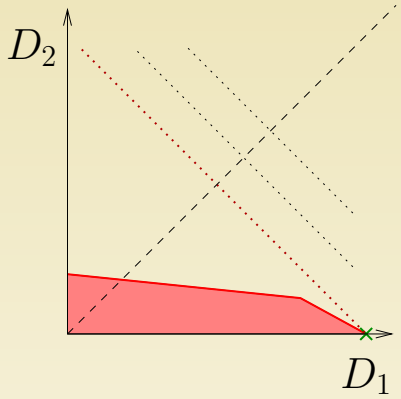
$$\text{MAXIMIZE} \left(\sum_{k \in A} \ln D_k \right)$$

Defining Fairness for our framework

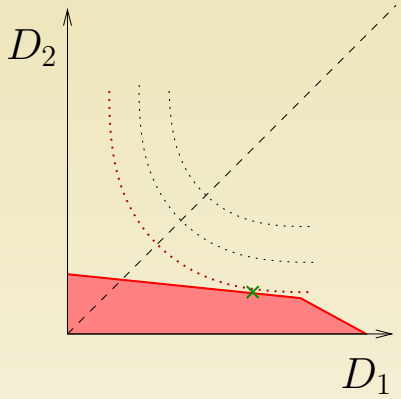
- Types d'équité:



Max min
$\min_k D_k$



Social welfare
$\sum_k D_k$



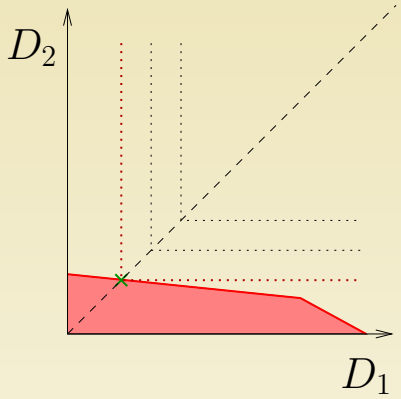
Proportional
$\prod_k D_k$

- A few problems with max-min fairness:
  - Not efficient when applications are very different
  - Seems to be hard to reach on such platforms [rr2005-45]
- Let's try proportional fairness!

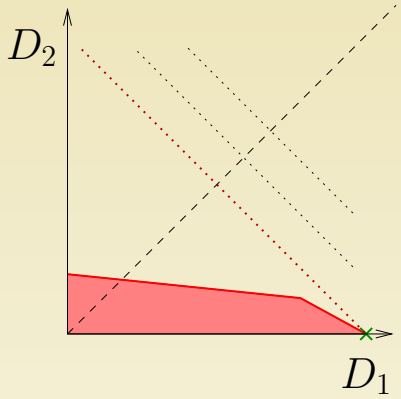
$$\text{MAXIMIZE} \left(\sum_{k \in A} \ln D_k \right)$$

Defining Fairness for our framework

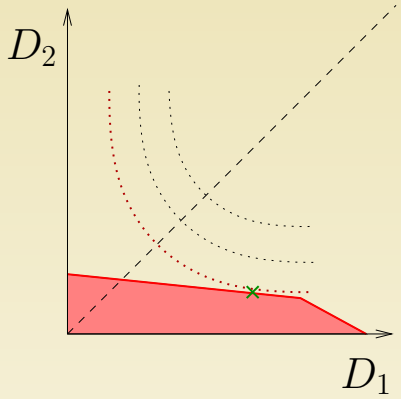
- Types d'équité:



Max min
$\min_k D_k$



Social welfare
$\sum_k D_k$



Proportional
$\prod_k D_k$

- A few problems with max-min fairness:
  - Not efficient when applications are very different
  - Seems to be hard to reach on such platforms [rr2005-45]
- Let's try proportional fairness!

$$\text{MAXIMIZE} \left(\sum_{k \in A} \ln D_k \right)$$

A new optimization problem

$$\text{MAXIMIZE} \quad f(\alpha) = \sum_{k \in A} \ln \left(\sum_{i \in P} \alpha_i^{(k)} \right)$$

$$\text{UNDER THE CONSTRAINTS} \quad \begin{cases} \forall i \in P : \sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} \leqslant W_i & \text{(Computations)} \\ \forall i \in P : \sum_{\substack{j \text{ tel que} \\ i \in M \to j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} \leqslant B_i & \text{(Communications)} \\ \forall i \in P, \forall k \in A : \alpha_i^{(k)} \geqslant 0 & \text{(Feasability)} \end{cases}$$

- Can be solved in polynomial time with semi-definite programming. It is very centralized though.

Can we solve it in a distributed way ?

Plan

Lagrangian Optimisation: basics

- Designed to solve non linear optimization problems:
  - Let $\alpha \to f(\alpha)$ be a function to maximize
  - Let $(C_i(\alpha) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints:
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geqslant 0, \text{ and } \alpha \geqslant 0 \end{cases}$$

- The Lagrangian function: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$
- The dual functionnal: $d(\lambda) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda)$
- Under some weak hypothesis, solving (P) is equivalent to solve the dual problem:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geqslant 0 \end{cases}$$

Lagrangian Optimisation: basics

- Designed to solve non linear optimization problems:
  - Let $\alpha \to f(\alpha)$ be a function to maximize
  - Let $(C_i(\alpha) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints:
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geqslant 0, \text{ and } \alpha \geqslant 0 \end{cases}$$

- The Lagrangian function: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$
- The dual functionnal: $d(\lambda) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda)$
- Under some weak hypothesis, solving (P) is equivalent to solve the dual problem:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geqslant 0 \end{cases}$$

Lagrangian Optimisation: basics

- Designed to solve non linear optimization problems:
  - Let $\alpha \to f(\alpha)$ be a function to maximize
  - Let $(C_i(\alpha) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints:
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geqslant 0, \text{ and } \alpha \geqslant 0 \end{cases}$$

- The Lagrangian function: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$
- The dual functionnal: $d(\lambda) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda)$
- Under some weak hypothesis, solving (P) is equivalent to solve the dual problem:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geqslant 0 \end{cases}$$

Lagrangian Optimisation: basics

- Designed to solve non linear optimization problems:
  - Let $\alpha \to f(\alpha)$ be a function to maximize
  - Let $(C_i(\alpha) \geqslant 0)_{i \in [1..n]}$ be a set of n constraints:
  - We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geqslant 0, \text{ and } \alpha \geqslant 0 \end{cases}$$

- The Lagrangian function: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$
- The dual functionnal: $d(\lambda) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda)$
- Under some weak hypothesis, solving (P) is equivalent to solve the dual problem:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geqslant 0 \end{cases}$$

Lagrangian Optimisation: basics

- Designed to solve non linear optimization problems:

So what?..

- Two coupled problems with simple constraints.
- The structure of constraints is transposed to (D).
- This technique has been used successfully for network resource sharing.

- T
- The dual functionnal: $d(\lambda) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda)$
- Under some weak hypothesis, solving (P) is equivalent to solve the dual problem:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geqslant 0 \end{cases}$$

Plan

Trying to use Lagrangian optimization

- What does the Lagrangien function look like ?

$$\mathcal{L}(\alpha, \lambda, \mu) = f(\alpha) - \sum_{i \in P} \lambda_i \cdot \left(\sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} - W_i \right) - \sum_{i \in P} \mu_i \cdot \left(\sum_{\substack{j \text{ tel que} \\ i \in M \to j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} - B_i \right)$$

- Let's compute the dual function $d(\lambda, \mu) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda, \mu)$:

$$\frac{\partial \mathcal{L}}{\partial \alpha_i^{(k)}} = 0 \Rightarrow \frac{1}{D_k} = \min_{i \in P} \left(w^{(k)} \lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j \right) = \min_{i \in P} \Pi(i, k)$$

Trying to use Lagrangian optimization

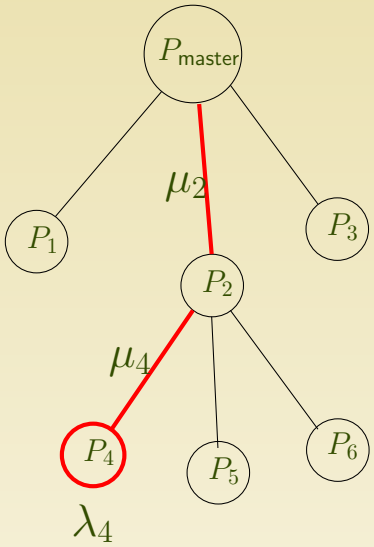
- What does the Lagrangien function look like ?

$$\mathcal{L}(\alpha, \lambda, \mu) = f(\alpha) - \sum_{i \in P} \lambda_i \cdot \left(\sum_{k \in A} \alpha_i^{(k)} \cdot w^{(k)} - W_i \right) - \sum_{i \in P} \mu_i \cdot \left(\sum_{\substack{j \text{ tel que} \\ i \in M \to j}} \sum_{k \in A} \alpha_j^{(k)} \cdot b^{(k)} - B_i \right)$$

- Let's compute the dual function $d(\lambda, \mu) = \max_{\alpha \geqslant 0} \mathcal{L}(\alpha, \lambda, \mu)$:

$$\frac{\partial \mathcal{L}}{\partial \alpha_i^{(k)}} = 0 \Rightarrow \frac{1}{D_k} = \min_{i \in P} \left(w^{(k)} \lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j \right) = \min_{i \in P} \Pi(i, k)$$

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j$

  can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

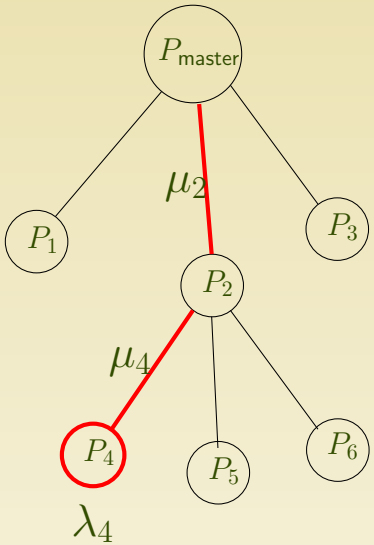
$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j$
  can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

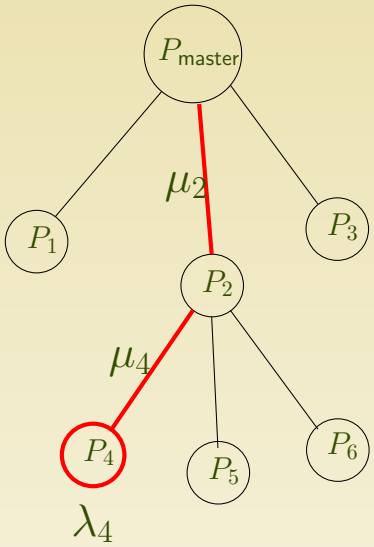
$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j$
  can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

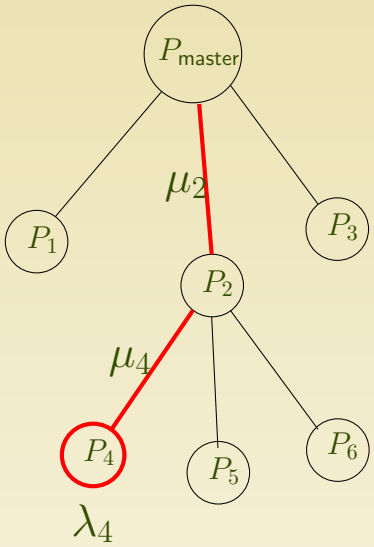
$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que } \mu_j \\ j \in M \to i}}$ can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

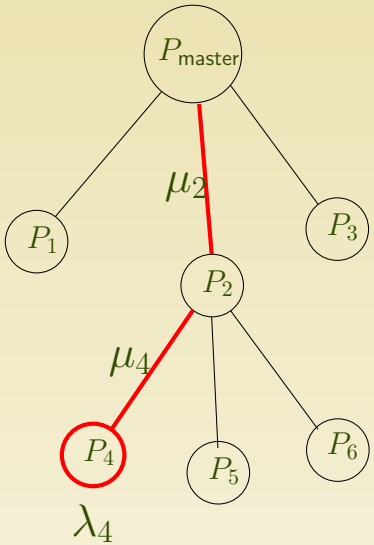
$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j$ can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

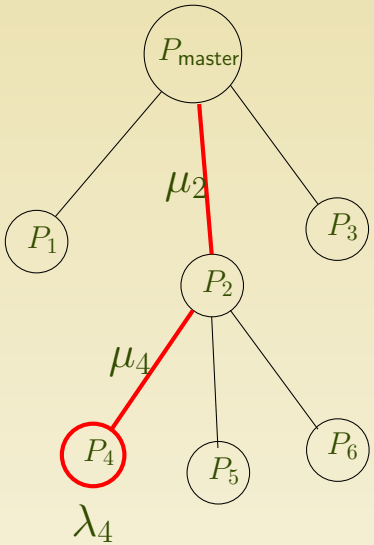
$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Trying to use Lagrangian optimization (cont'd)



- $\Pi(i,k) = w^{(k)}\lambda_i + b^{(k)} \sum_{\substack{j \text{ tel que} \\ j \in M \to i}} \mu_j$
  can be seen as the price application k has to pay to access P_i. The *best deal* is thus:

$$\Pi(k) = \min_{i \in P} \Pi(i,k)$$

- What does the dual function look like?

$$d(\lambda,\mu) = \sum_{i \in P} \big(\lambda_i W_i + \mu_i B_i\big) - \sum_{k \in A} \ln\big(\Pi(k)\big) - cte$$

- d is convex. Distributed gradient methods can be used to minimize d: each resource will update it's price accordingly.
- Minimizing d can be seen as maximizing the global *income* of resources.
- The whole process can hence be seen as a bargain between applications and resources.

Computing the prices

- Sketch of the algorithm:
  - Each worker gives *its* best local deal (depending on its own λ and μ and on the best deals of its own workers) to *its* master.
  - Best deals are thus propagated to the *root master* along the tree
  - For each application, each master informs its workers that have the best deal that he wishes to use them.
  - Best workers are thus selected along the tree.
  - Each worker updates its price depending on applications that want to use its resource.
- Main difficulties:
  - The Lagrangian function is not C^2 $\Rightarrow$ there are oscillations around angles.
  - $\ln$ is not defined in 0. Step-size should thus be carefully chosen...

Computing the prices

- Sketch of the algorithm:
  - Each worker gives *its* best local deal (depending on its own λ and μ and on the best deals of its own workers) to *its* master.
  - Best deals are thus propagated to the *root master* along the tree
  - For each application, each master informs its workers that have the best deal that he wishes to use them.
  - Best workers are thus selected along the tree.
  - Each worker updates its price depending on applications that want to use its resource.
- Main difficulties:
  - The Lagrangian function is not $C^2 \Rightarrow$ there are oscillations around angles.
  - $\ln$ is not defined in 0. Step-size should thus be carefully chosen...

Computing the prices

- Sketch of the algorithm:
  - Each worker gives *its* best local deal (depending on its own λ and μ and on the best deals of its own workers) to *its* master.
  - Best deals are thus propagated to the *root master* along the tree
  - For each application, each master informs its workers that have the best deal that he wishes to use them.
  - Best workers are thus selected along the tree.
  - Each worker updates its price depending on applications that want to use its resource.
- Main difficulties:
  - The Lagrangian function is not $C^2 \Rightarrow$ there are oscillations around angles.
  - $\ln$ is not defined in 0. Step-size should thus be carefully chosen...

Computing the prices

- Sketch of the algorithm:
  - Each worker gives *its* best local deal (depending on its own λ and μ and on the best deals of its own workers) to *its* master.
  - Best deals are thus propagated to the *root master* along the tree
  - For each application, each master informs its workers that have the best deal that he wishes to use them.
  - Best workers are thus selected along the tree.
  - Each worker updates its price depending on applications that want to use its resource.
- Main difficulties:
  - The Lagrangian function is not C^2 $\Rightarrow$ there are oscillations around angles.
  - $\ln$ is not defined in 0. Step-size should thus be carefully chosen...

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Coming back to our original problem

- We found the optimal deal ($\Pi(k)$) and prices (λ's and μ's) but we still don't have the optimal values of the $\alpha_i^{(k)}$'s.
- However, we have the throughput of each application: $D_k = \Pi(k)$
- Given feasible values for the D_k's, how can we find valid values for the $\alpha_i^{(k)}$'s ?...
- ...Using Olivier's adaptation of the Awerbuch Leighton algorithm (originally designed for the multi-commodity flow problem)!
- Our original problem is decomposed in 2 sub-problems:
  - Solving the dual problem: determining optimal prices (λ, μ)
  - Finding valid rates $\alpha_i^{(k)}$ from the prices.
- Both problems are optimally solved (or approximated) with a distributed algorithm!

Plan

SimGrid, MSG, plates-formes tiers

- The simulator is SimGrid.
- The platform generator is Tiers, but very simple trees here.
- Fully synchronous gradient.
- Checking the correctness of the results using semi-definite programming.

SimGrid, MSG, plates-formes tiers

- The simulator is SimGrid.
- The platform generator is Tiers, but very simple trees here.
- Fully synchronous gradient.
- Checking the correctness of the results using semi-definite programming.

SimGrid, MSG, plates-formes tiers

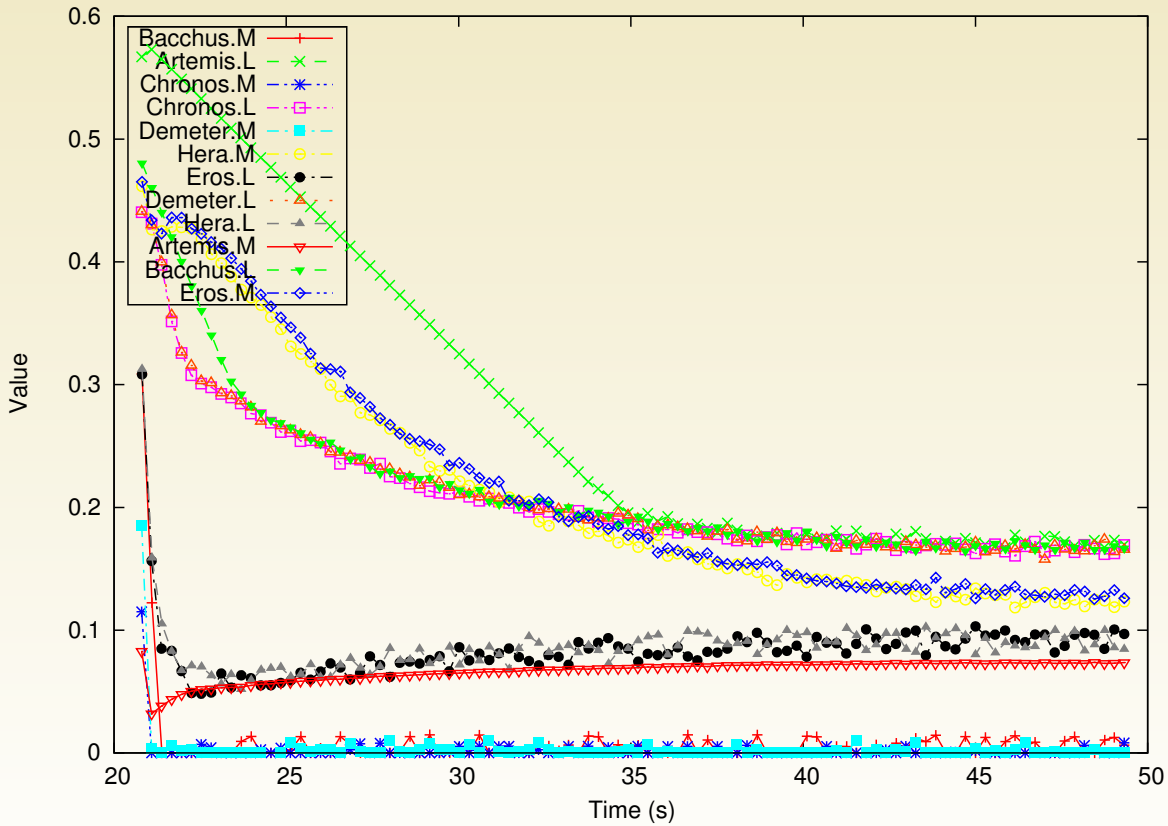
- The simulator is SimGrid.
- The platform generator is Tiers, but very simple trees here.
- Fully synchronous gradient.
- Checking the correctness of the results using semi-definite programming.

SimGrid, MSG, plates-formes tiers

- The simulator is SimGrid.
- The platform generator is Tiers, but very simple trees here.
- Fully synchronous gradient.
- Checking the correctness of the results using semi-definite programming.

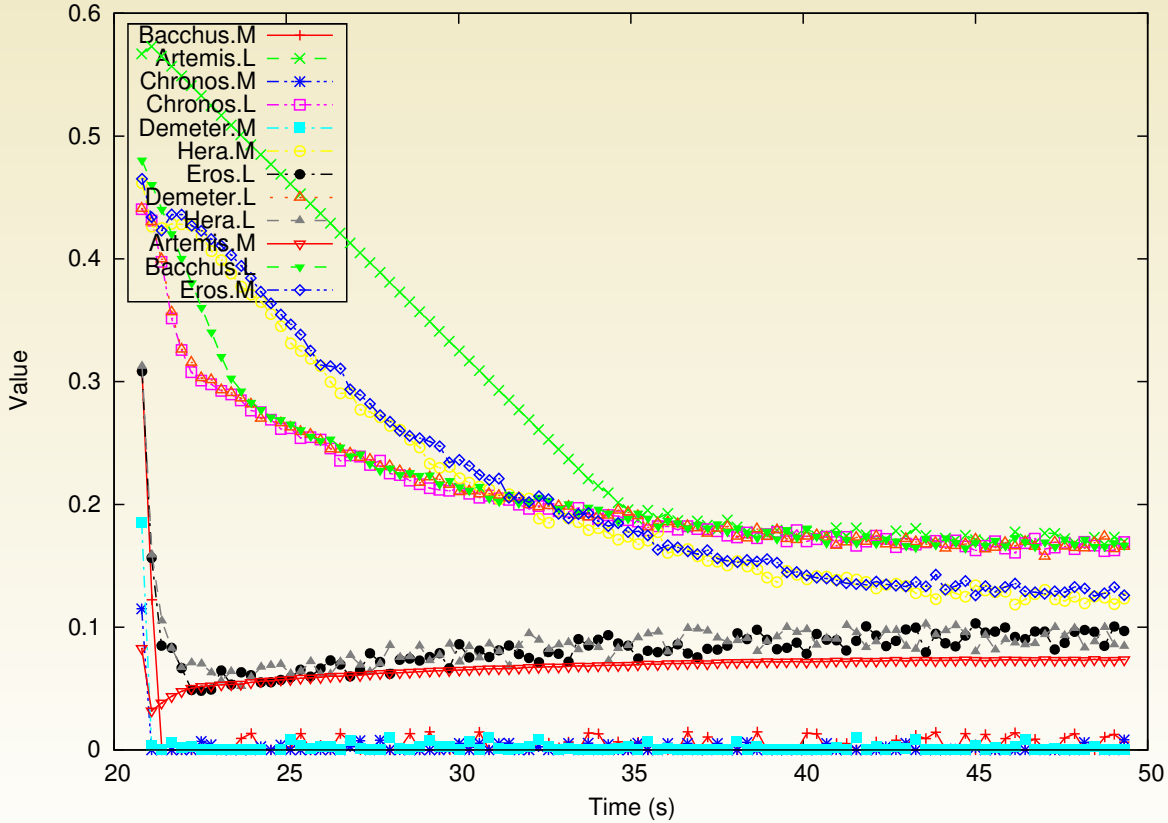
Convergence

- 6 processors, 2 applications
- Trajectories *merge*

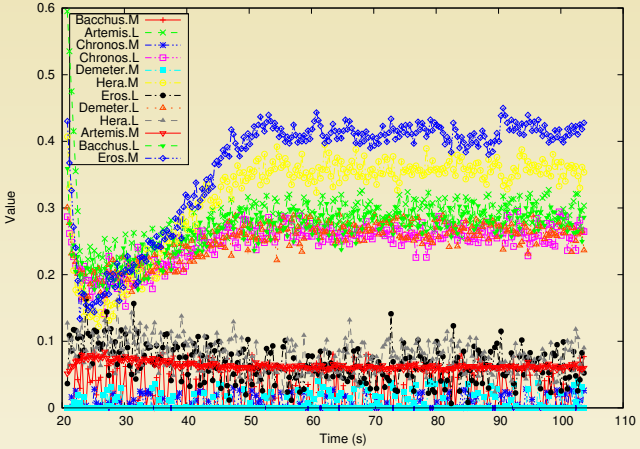


Convergence

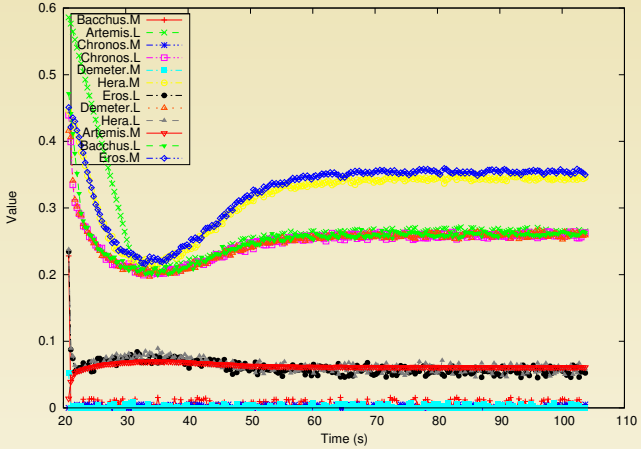
- 6 processors, 2 applications
- Trajectories *merge*



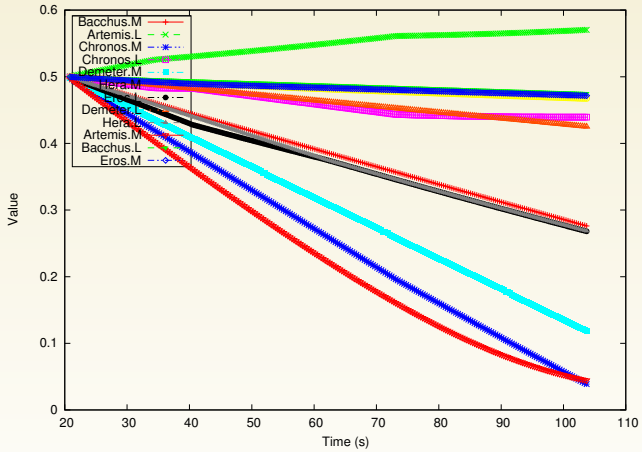
Oscillations: choosing the right step



(a) $\sigma = 1.5.10^{-2}$



(b) $\sigma = 3.10^{-3}$



(c) $\sigma = 1.0.10^{-5}$

Plan

Perpectives

- Speeding up the bargain
  - Trying asynchronous steps.
  - Smoothing the min.
  - Studying the ability to react to platform changes.
- Extending to general platforms with tree deployments is trivial.
- Trying other fairness measures.
- Implementing the Awerbuch Leighton algorithm and coupling with the bargain $\rightsquigarrow$ will it work in practice ?

Perpectives

- Speeding up the bargain
  - Trying asynchronous steps.
  - Smoothing the min.
  - Studying the ability to react to platform changes.
- Extending to general platforms with tree deployments is trivial.
- Trying other fairness measures.
- Implementing the Awerbuch Leighton algorithm and coupling with the bargain $\rightsquigarrow$ will it work in practice ?

Perpectives

- Speeding up the bargain
  - Trying asynchronous steps.
  - Smoothing the min.
  - Studying the ability to react to platform changes.
- Extending to general platforms with tree deployments is trivial.
- Trying other fairness measures.
- Implementing the Awerbuch Leighton algorithm and coupling with the bargain $\rightsquigarrow$ will it work in practice ?

Perpectives

- Speeding up the bargain
  - Trying asynchronous steps.
  - Smoothing the min.
  - Studying the ability to react to platform changes.
- Extending to general platforms with tree deployments is trivial.
- Trying other fairness measures.
- Implementing the Awerbuch Leighton algorithm and coupling with the bargain $\rightsquigarrow$ will it work in practice ?

Bibliographie

O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert.
Scheduling multiple bags of tasks on heterogeneous master-worker platforms: centralized versus distributed solutions.
Research Report 2005-45, LIP, ENS Lyon, France, Sept. 2005.
Available at graal.ens-lyon.fr/~yrobert/rr2005-45.ps.