Distributed Deadlock-Avoidance

César Sánchez

IMDEA Software Institute, Spain

DRV Workshop, Bertinoro 19-May, 2016

Distributed Deadlock-A...ce

A little story about how static knowledge can help solve unsolvable problems

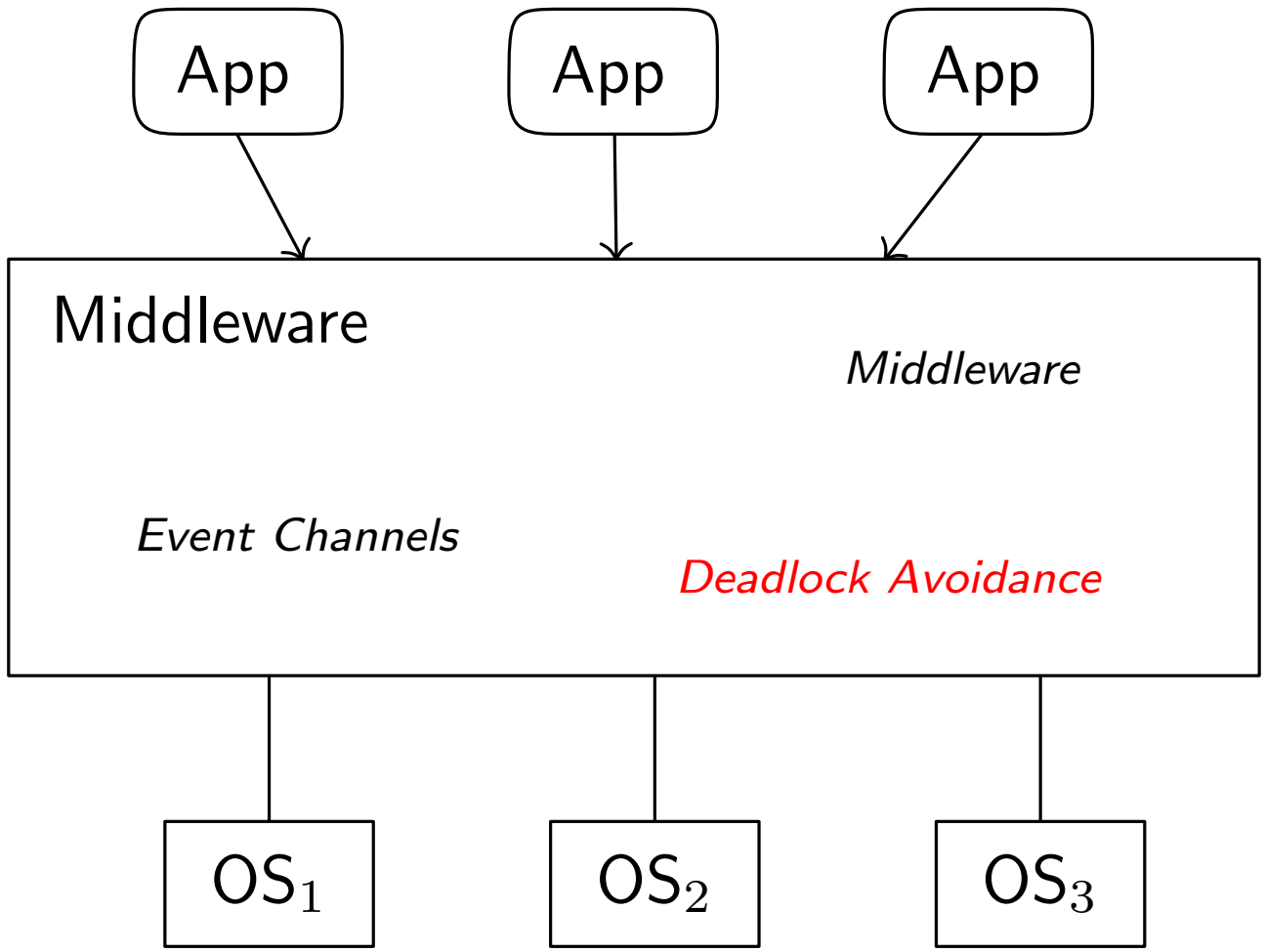
César Sánchez

IMDEA Software Institute, Spain

DRV Workshop, Bertinoro 19-May, 2016

Introduction

Goal: Formalization of middleware services



Deadlocks

Deadlock is one of the classical problems in CS

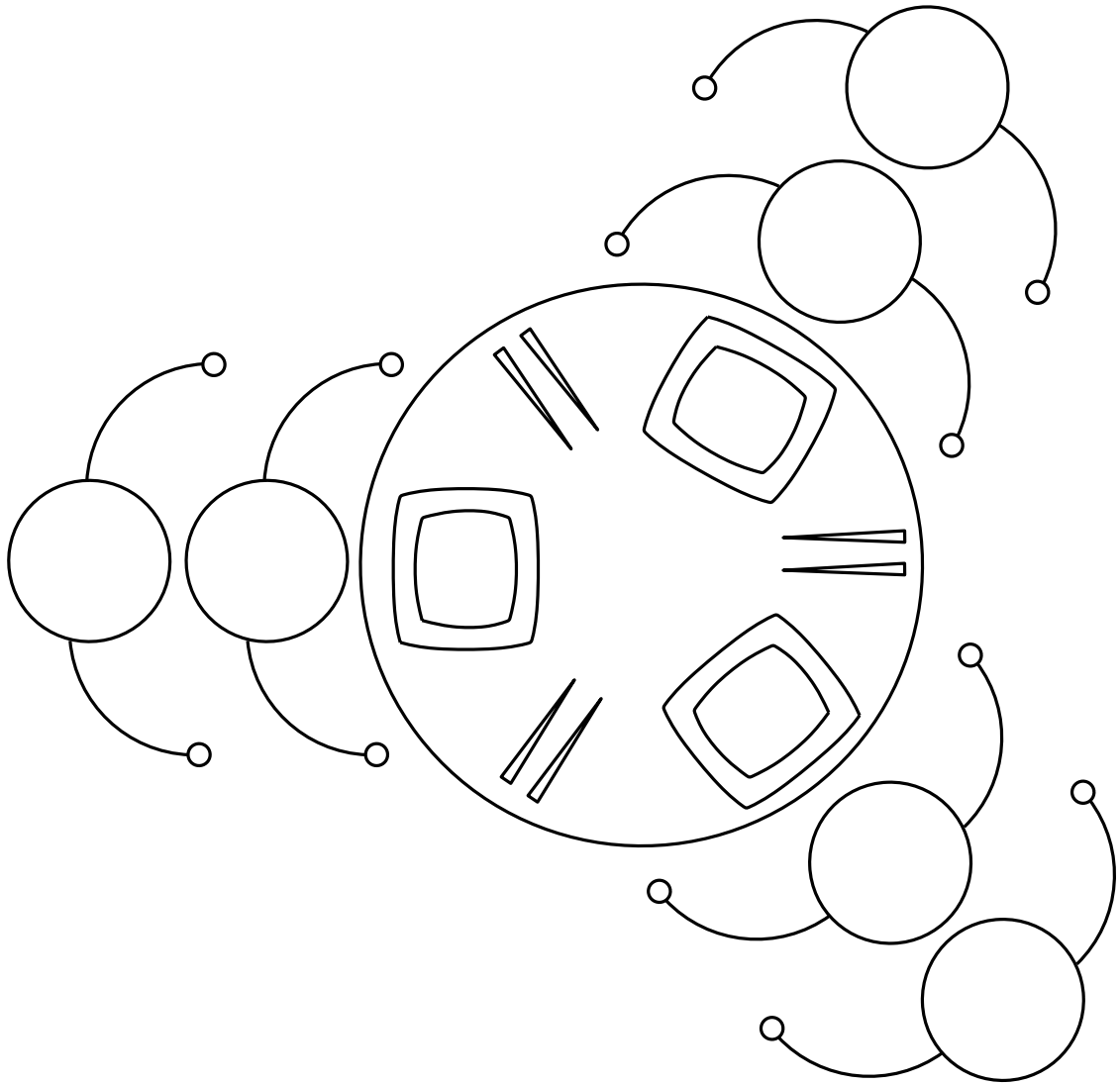
One (common) approach is the ostrich approach

The other approaches are: detection, prevention and avoidance.

	centralized	distributed
detection	OK	OK
prevention	OK	OK
avoidance	Banker's	impractical

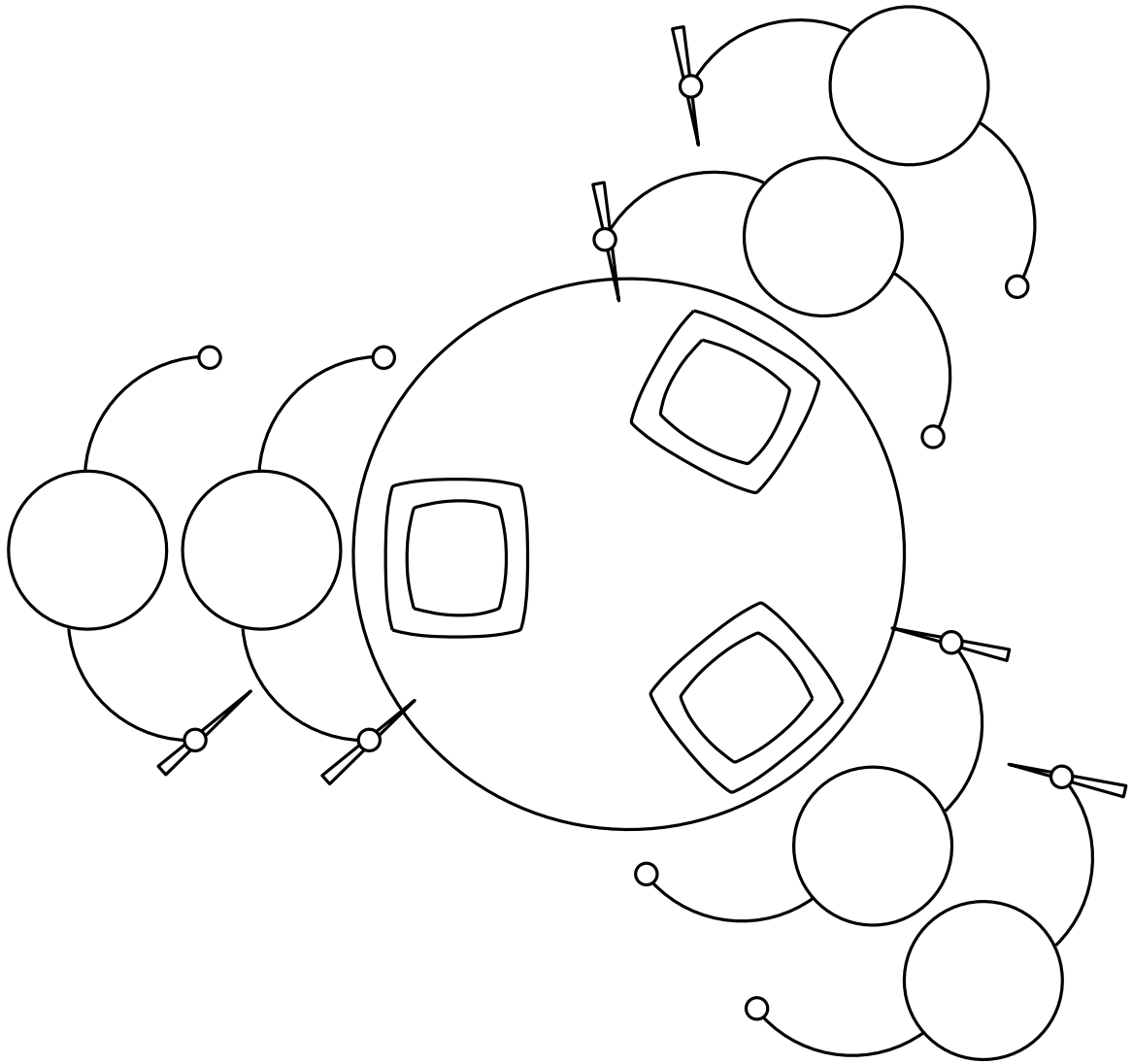
Efficient dynamic resource allocation can have a big practical impact.

Distributed Dinning Philosophers



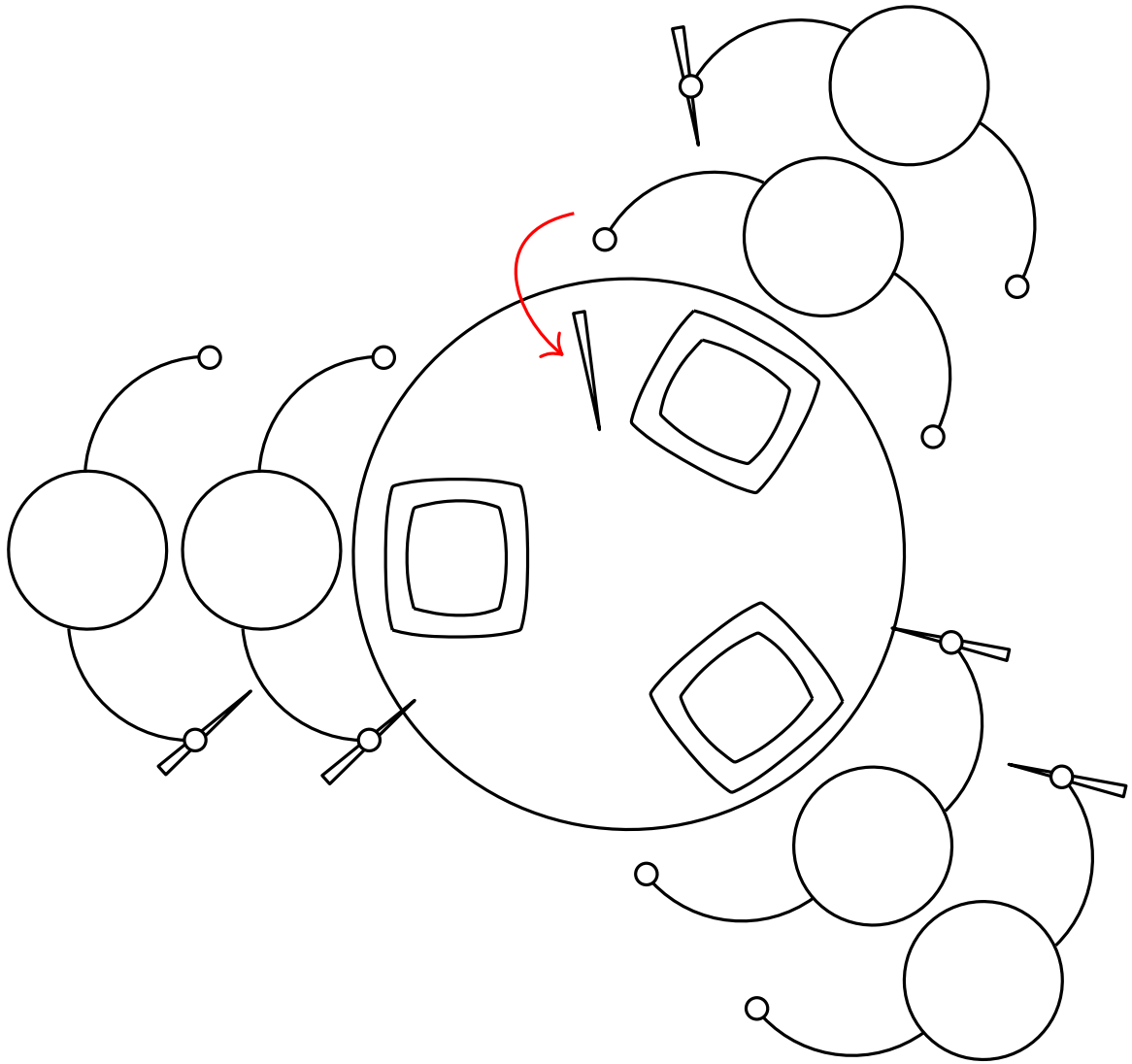
Distributed Dinning Philosophers

A deadlock state



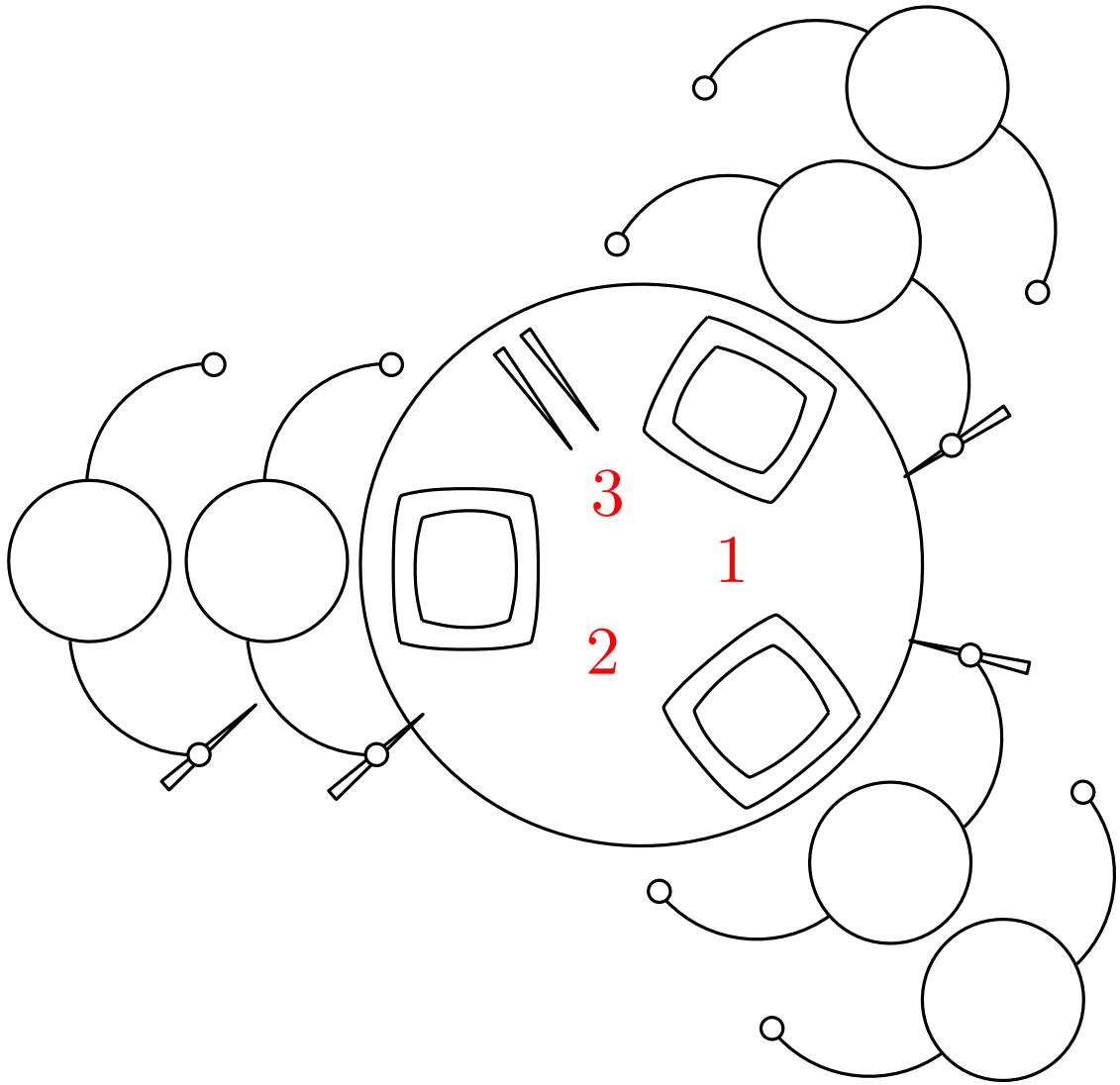
Distributed Dinning Philosophers

Detection:



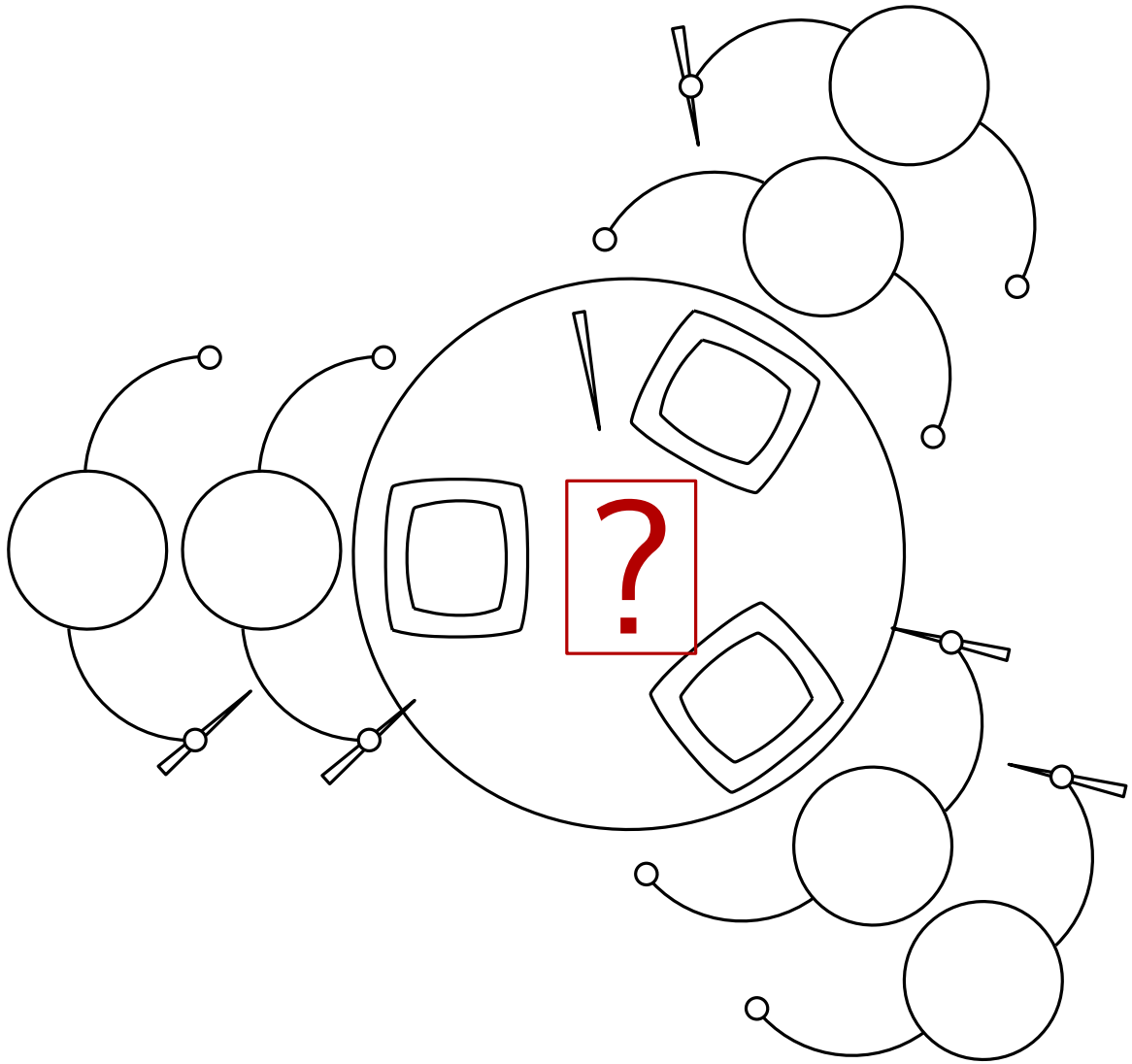
Distributed Dinning Philosophers

Prevention:



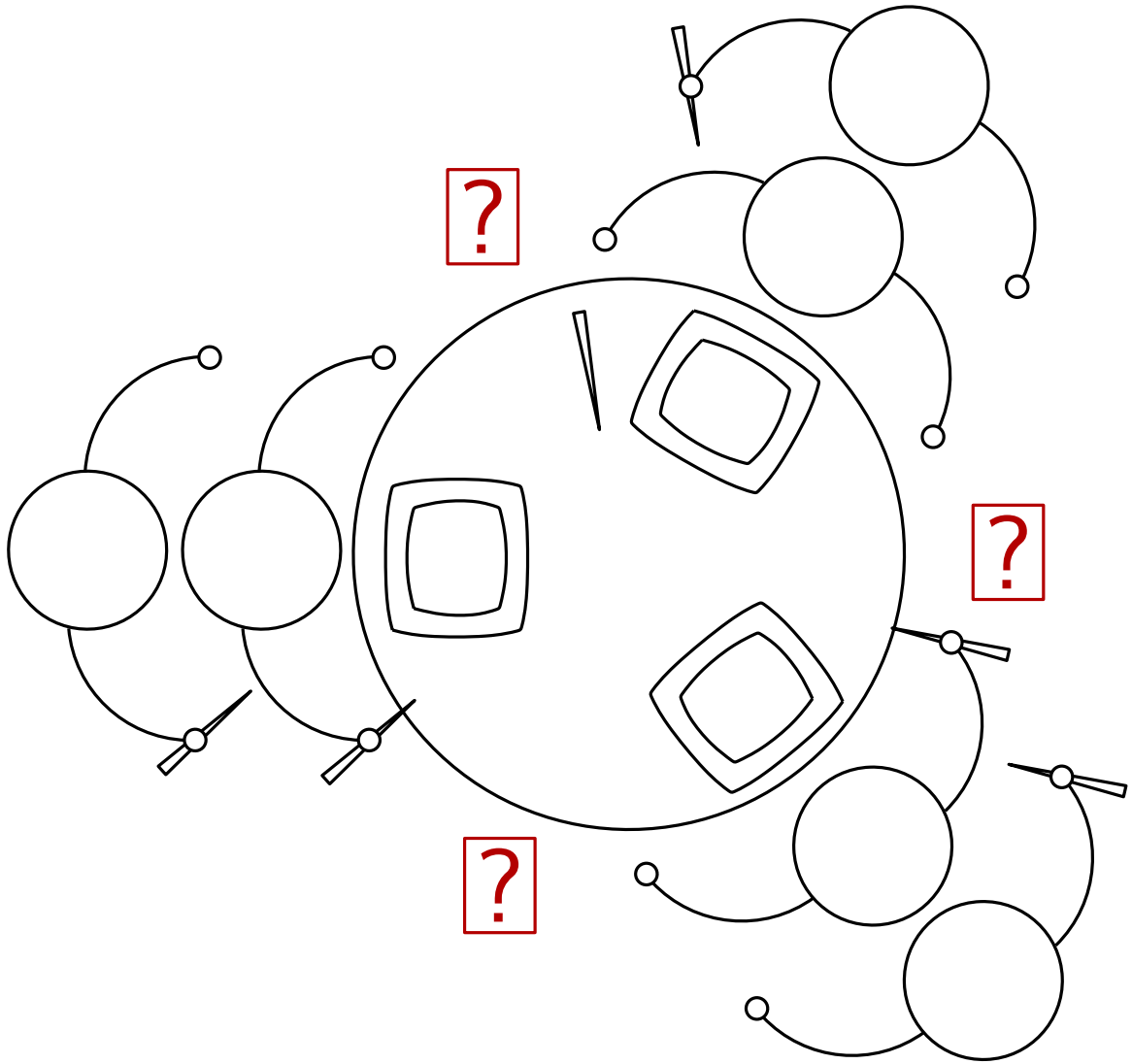
Distributed Dinning Philosophers

Avoidance:

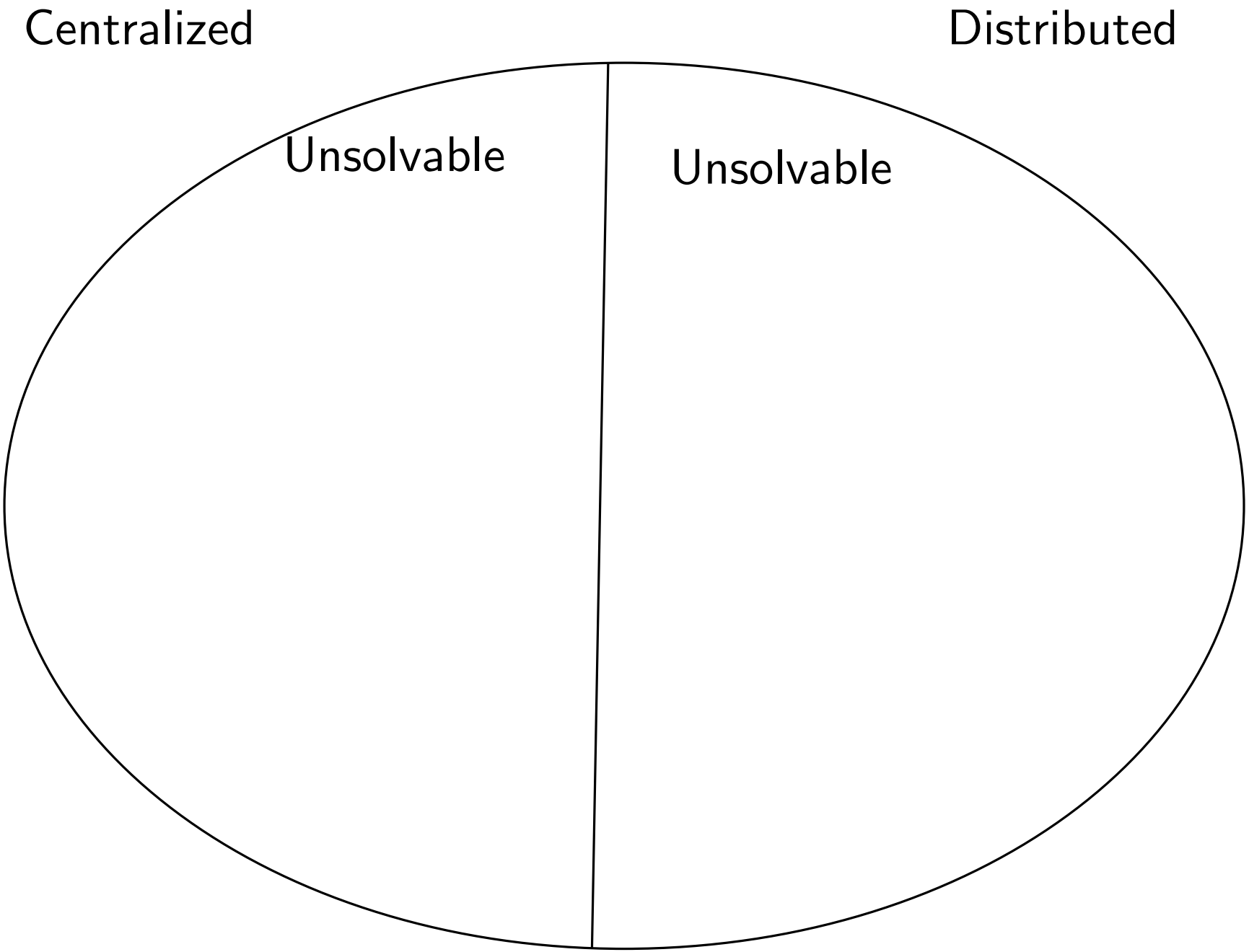


Distributed Dinning Philosophers

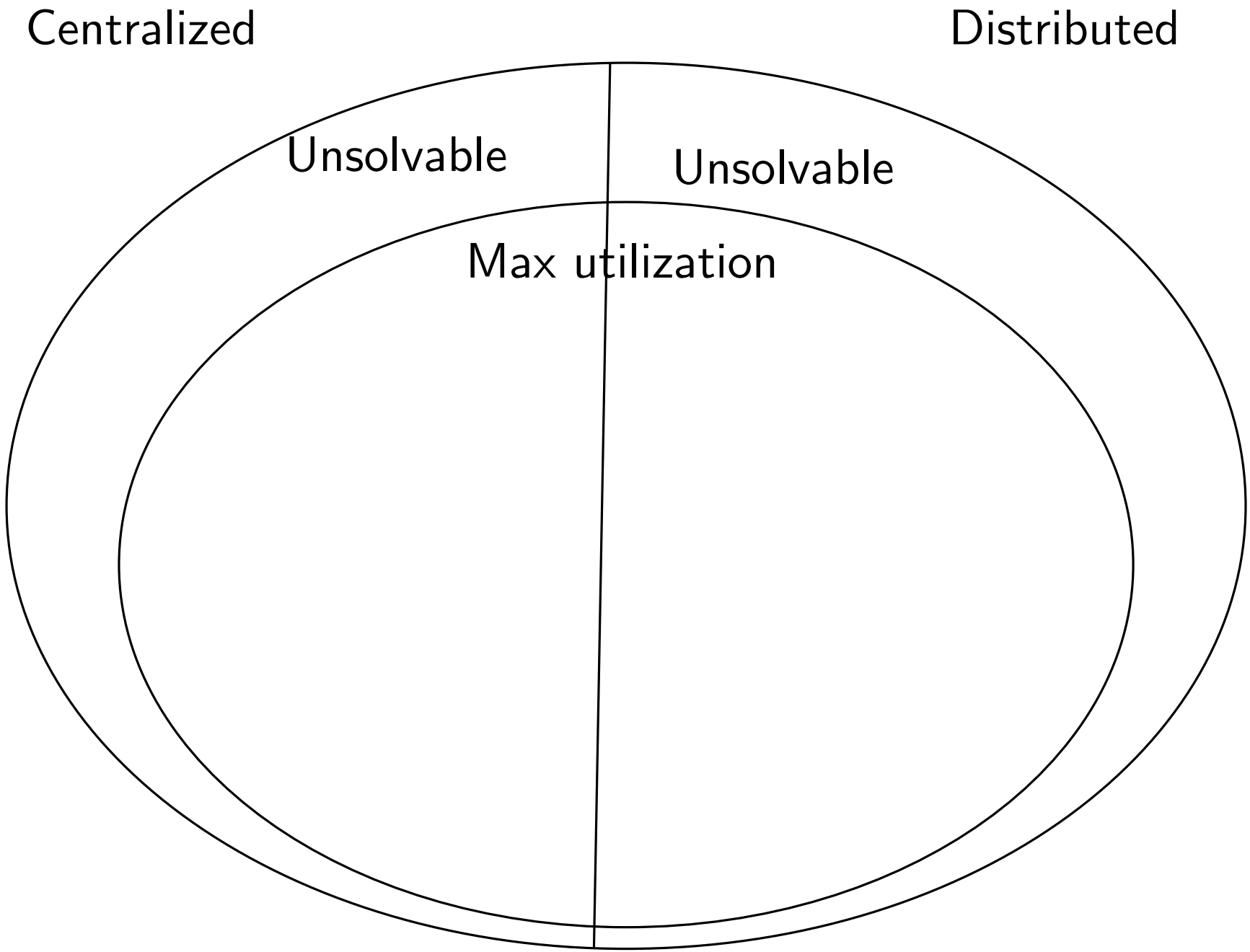
Distributed Avoidance:



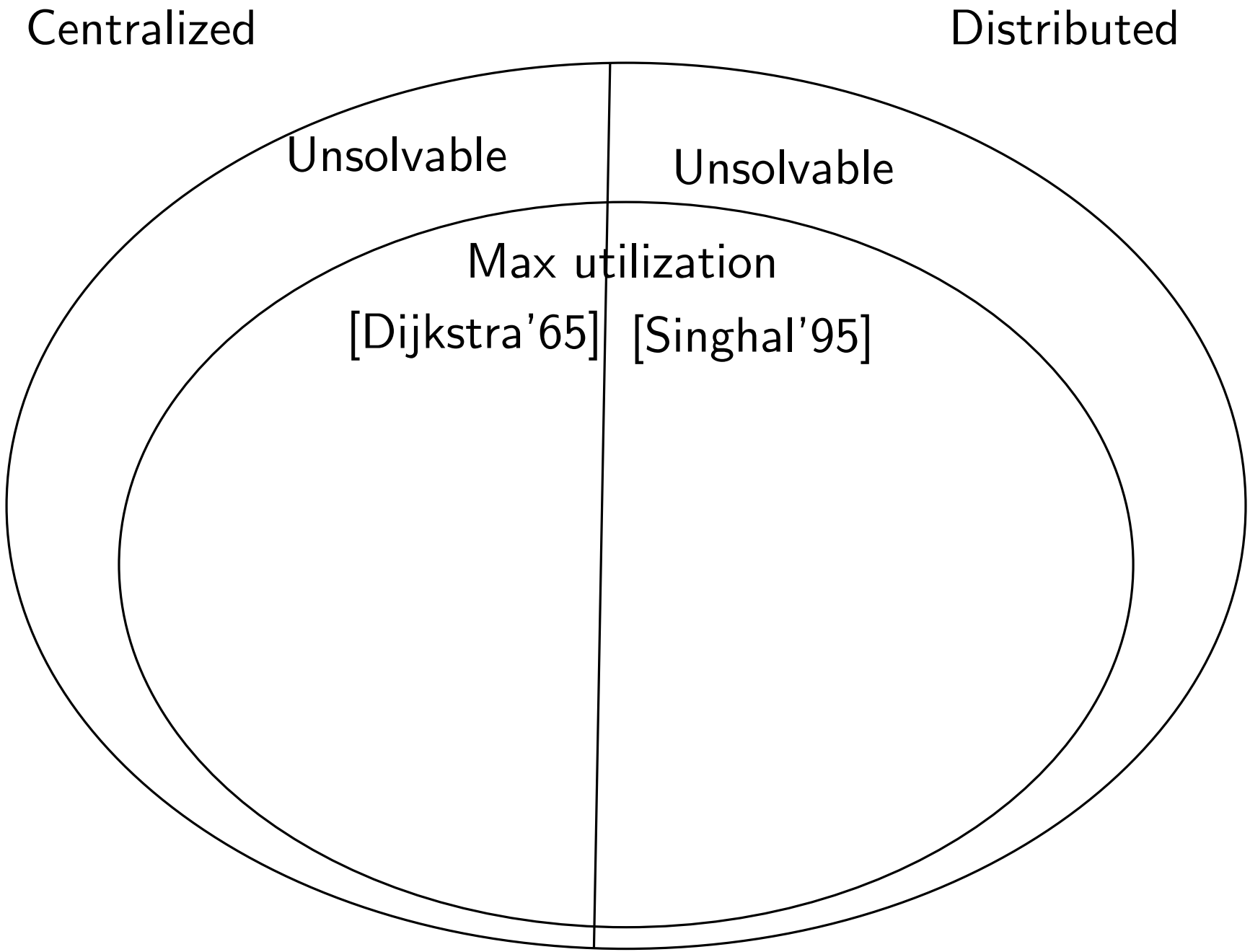
Deadlock Avoidance Problem Space



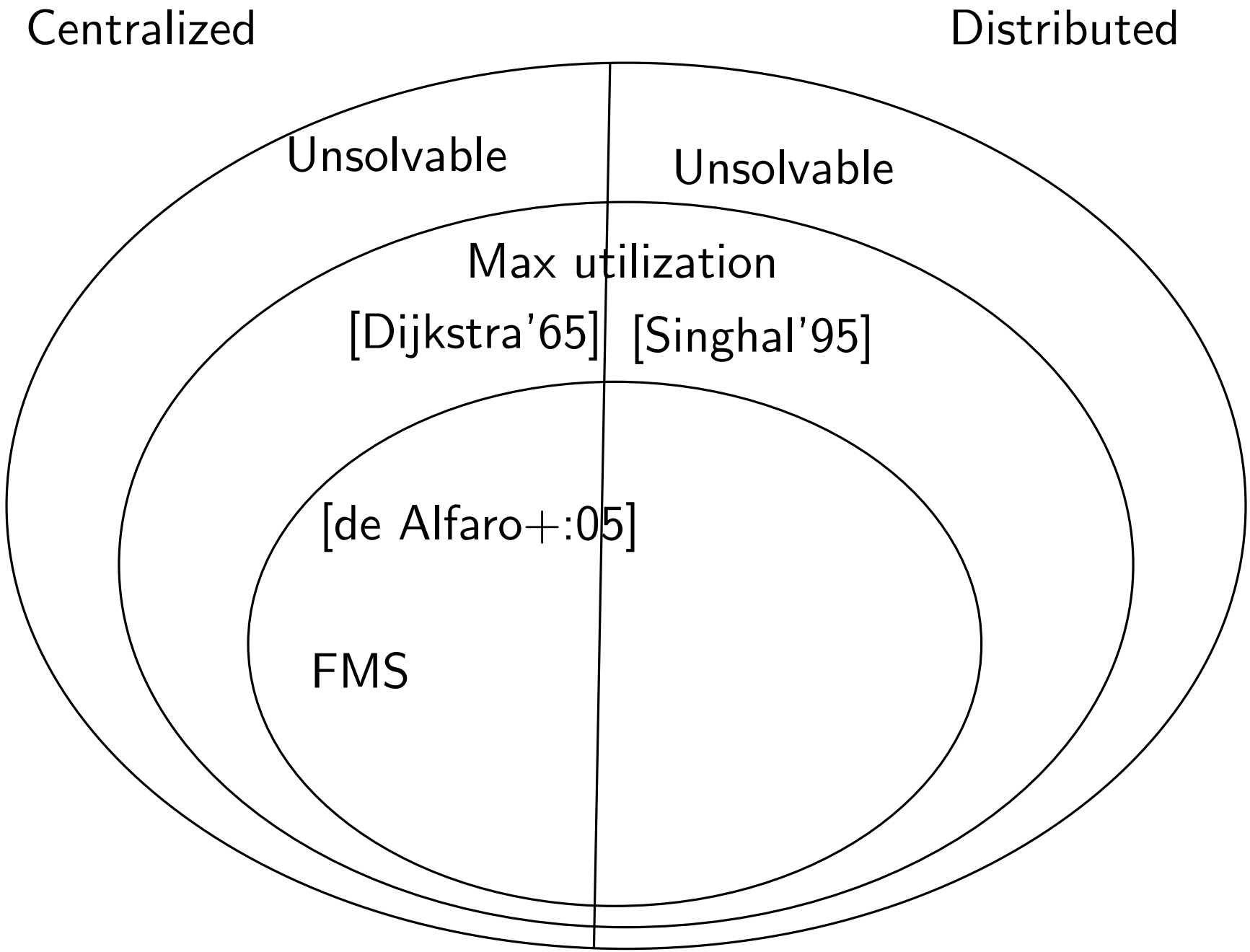
Deadlock Avoidance Problem Space



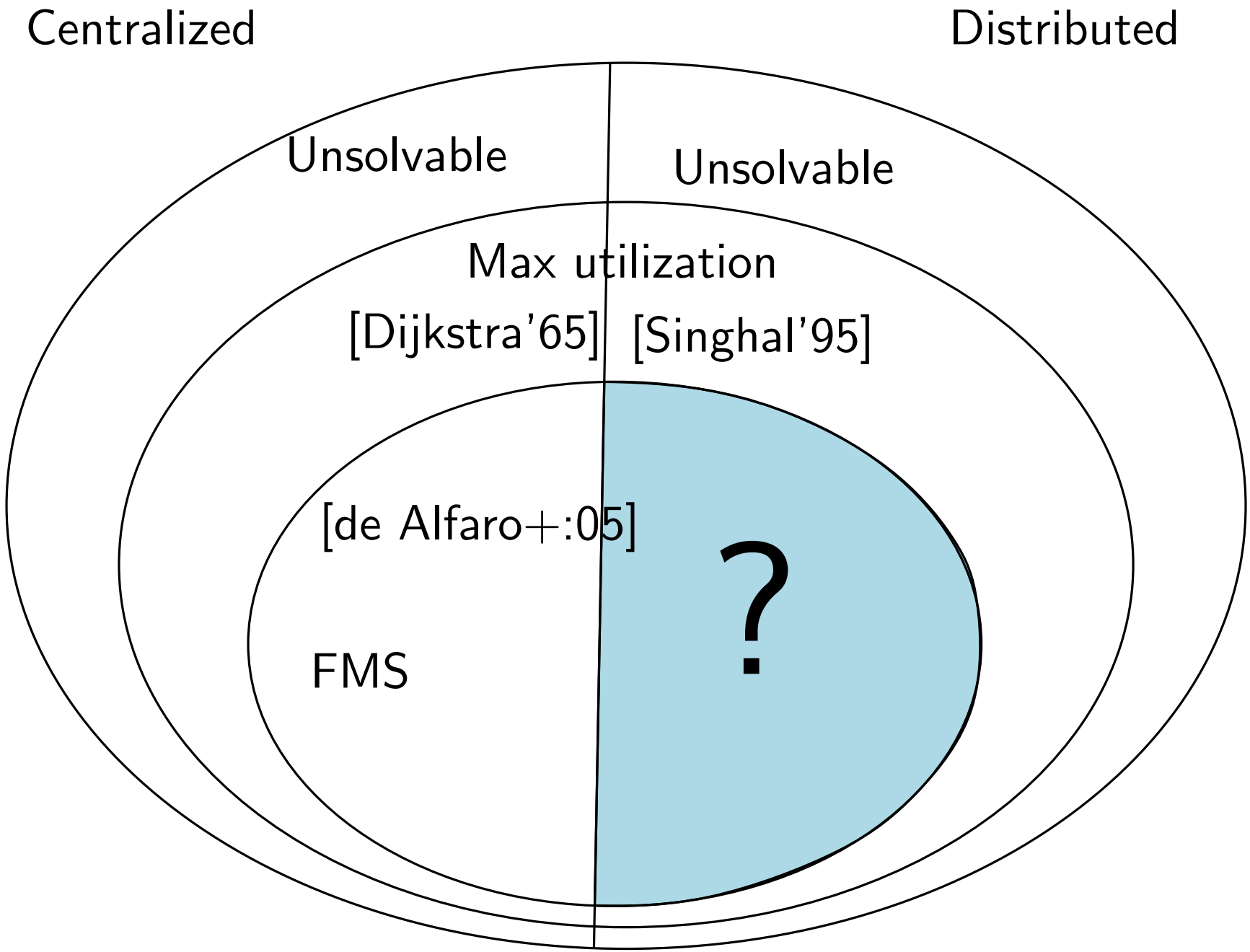
Deadlock Avoidance Problem Space



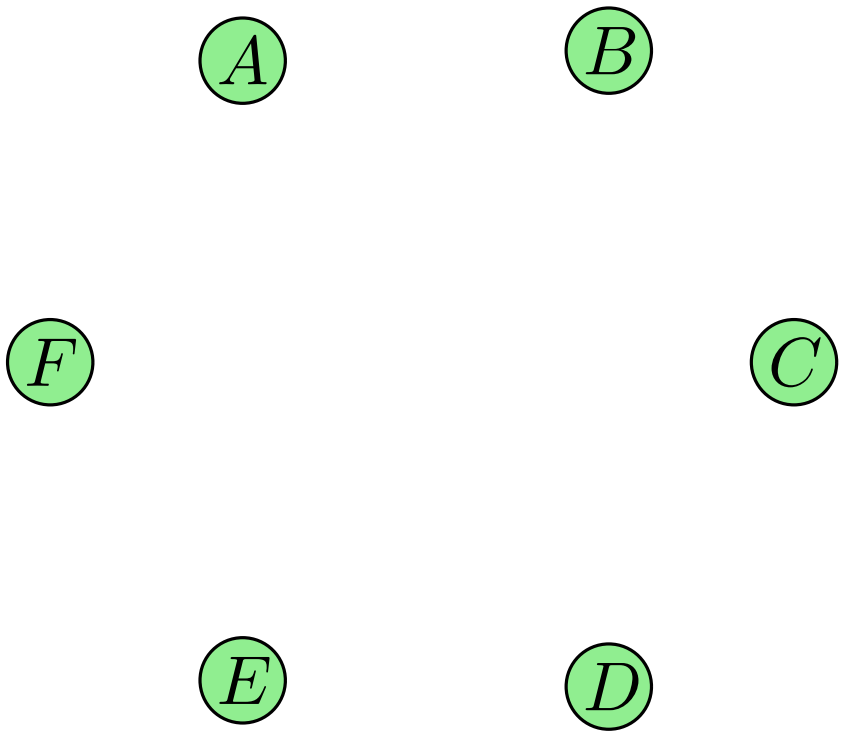
Deadlock Avoidance Problem Space



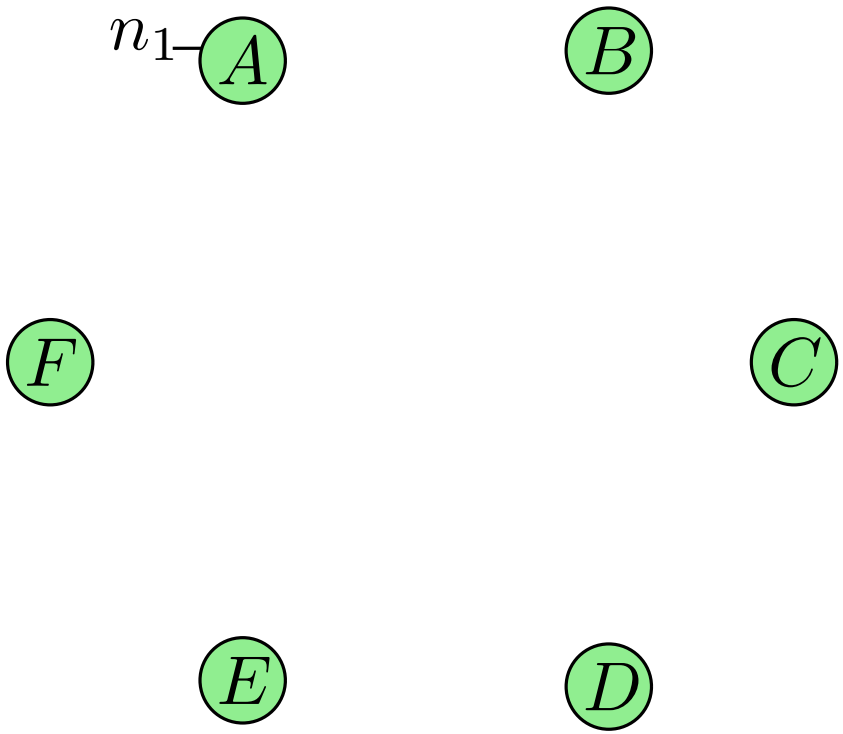
Deadlock Avoidance Problem Space



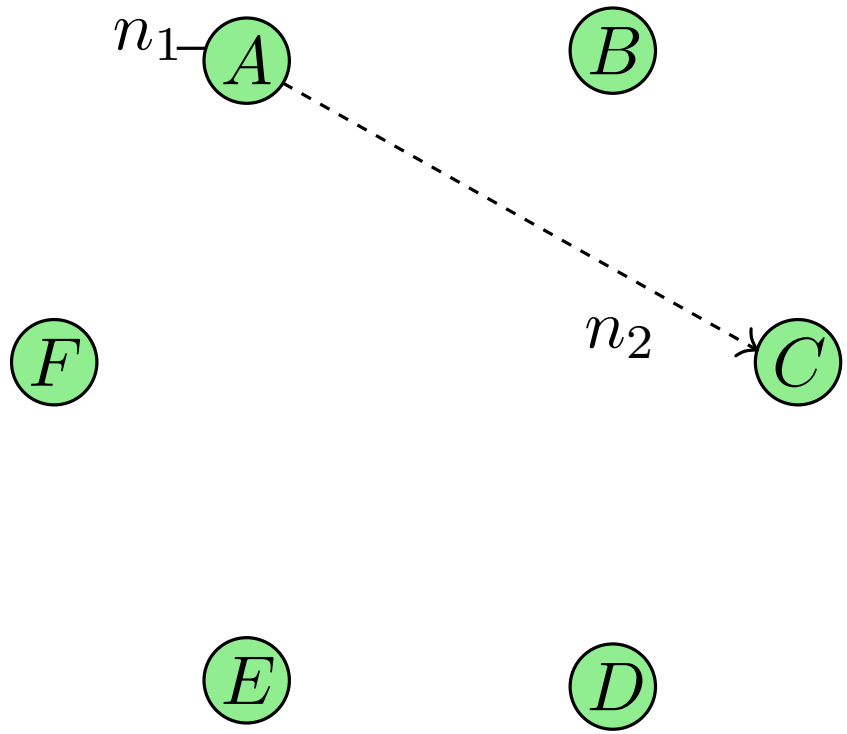
Distributed Real-Time and Embedded



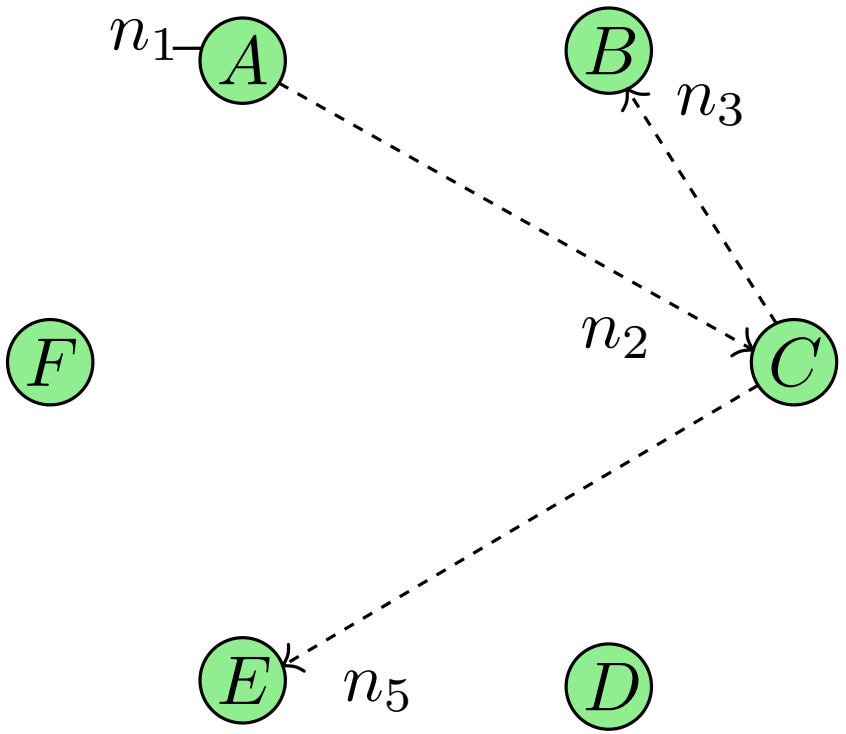
Distributed Real-Time and Embedded



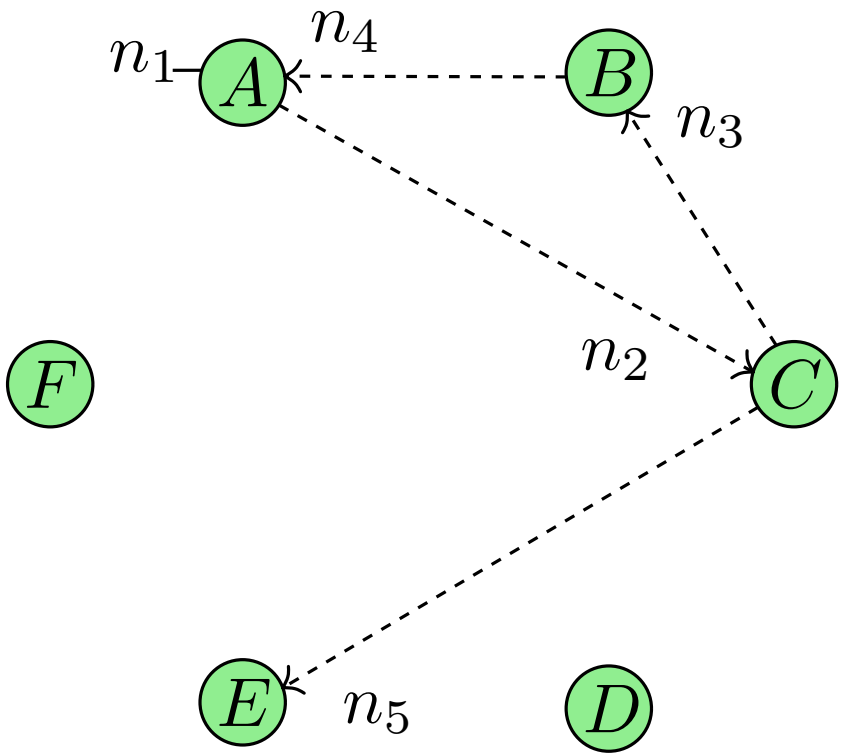
Distributed Real-Time and Embedded



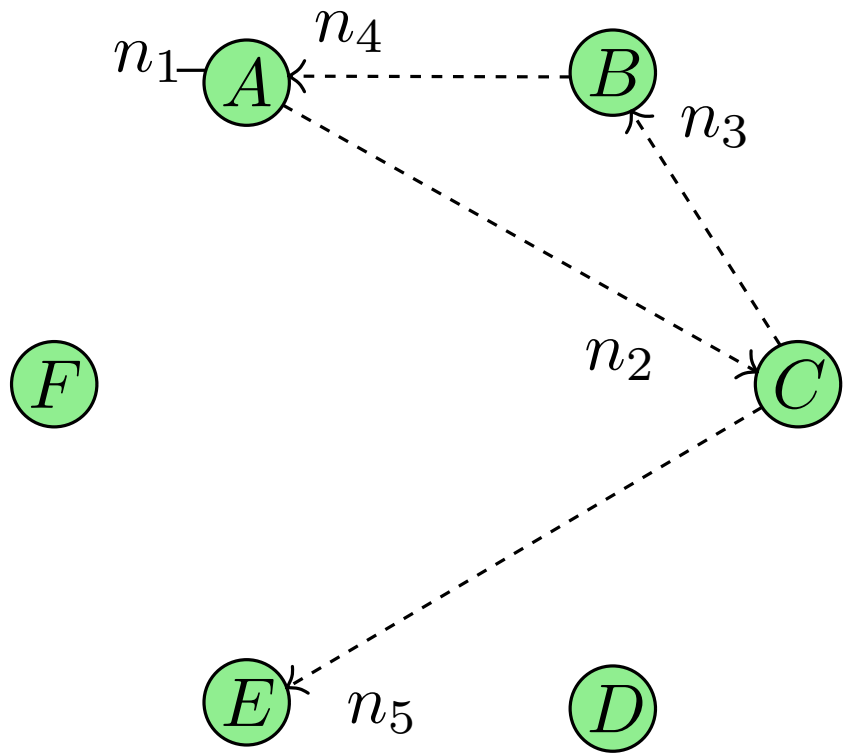
Distributed Real-Time and Embedded



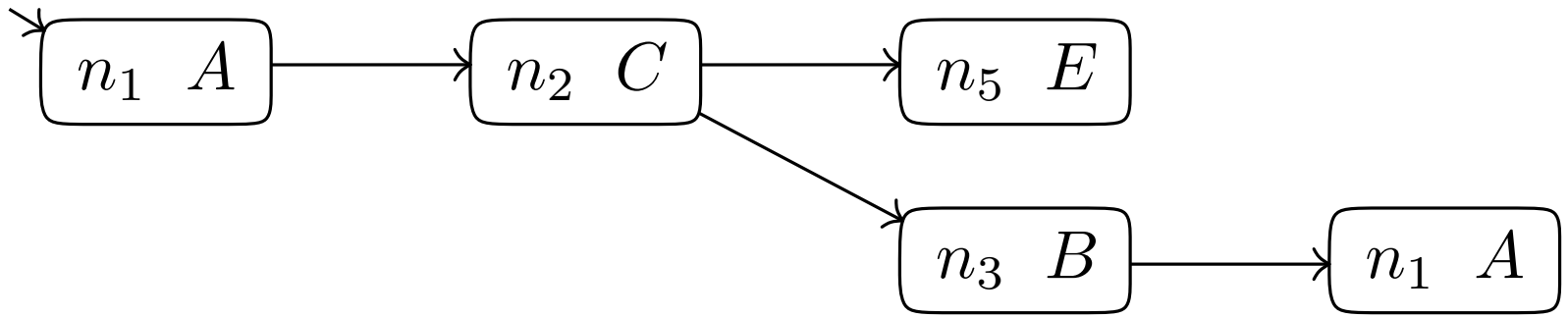
Distributed Real-Time and Embedded



Distributed Real-Time and Embedded



Sequence of calls:



Distributed Real-Time Embedded Systems

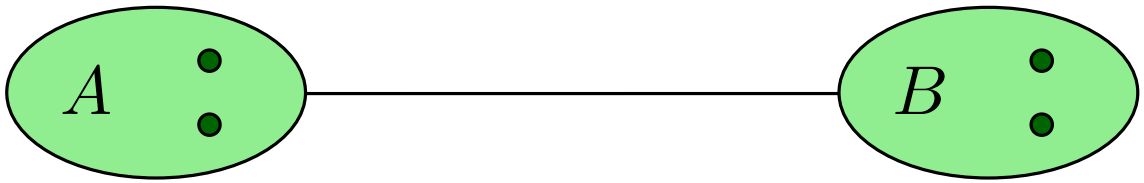
Distributed Real-Time Embedded Systems:

- Asynchronous distributed system
- Limited Resources
- *Wait-on-connection*
- Arbitrary number of processes spawned
- All processes terminate

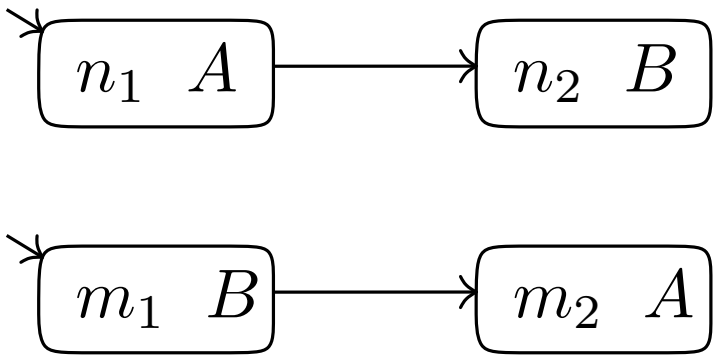
Problem: deadlocks are possible if no controller is used

Example of Deadlock

Two sites, with two resources each:

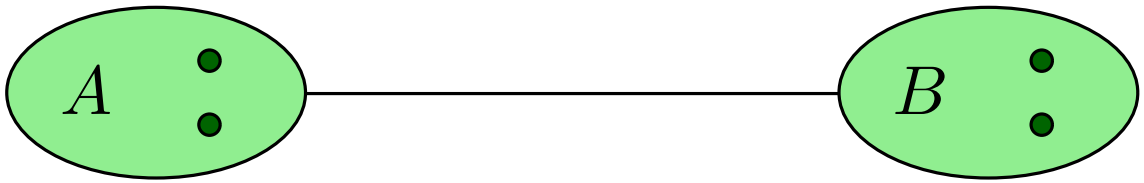


and the call graph:

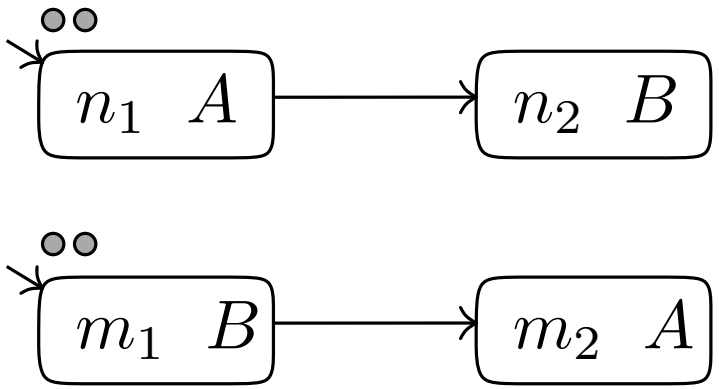


Example of Deadlock

Two sites, with two resources each:

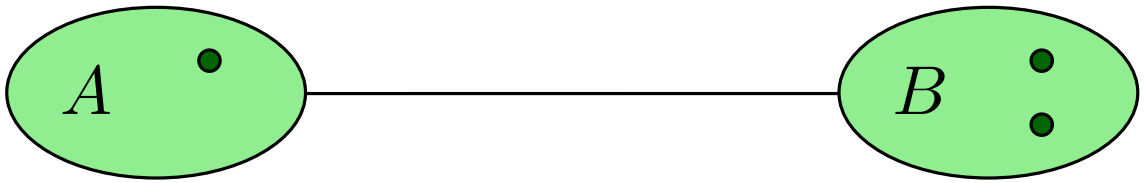


and the call graph:

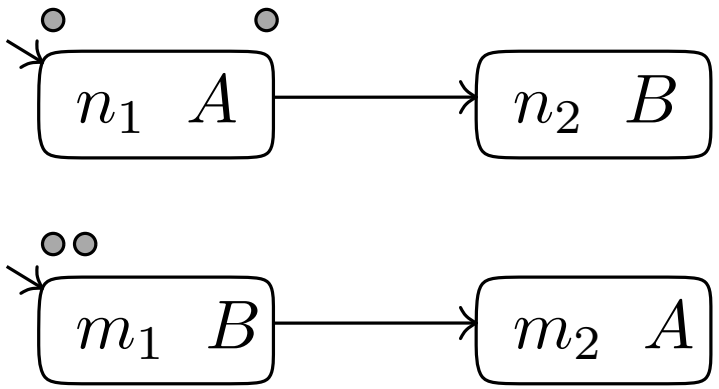


Example of Deadlock

Two sites, with two resources each:

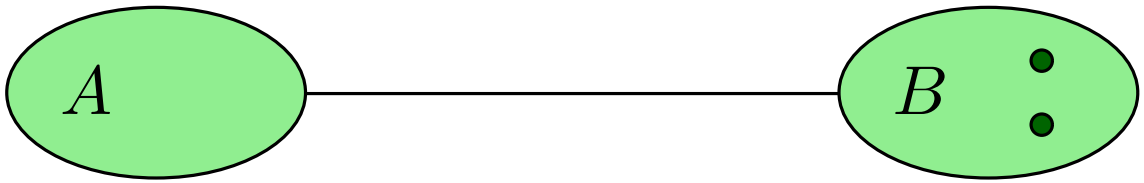


and the call graph:

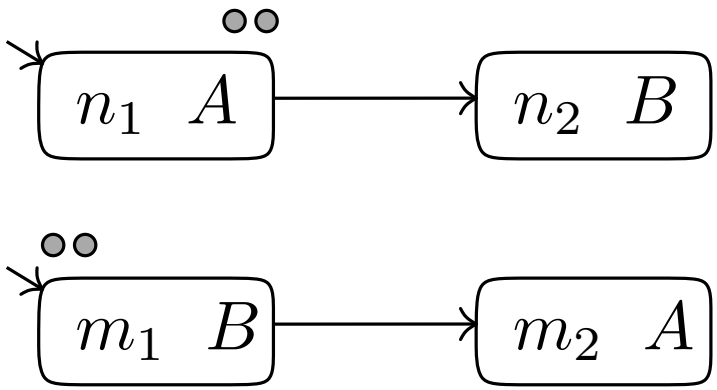


Example of Deadlock

Two sites, with two resources each:

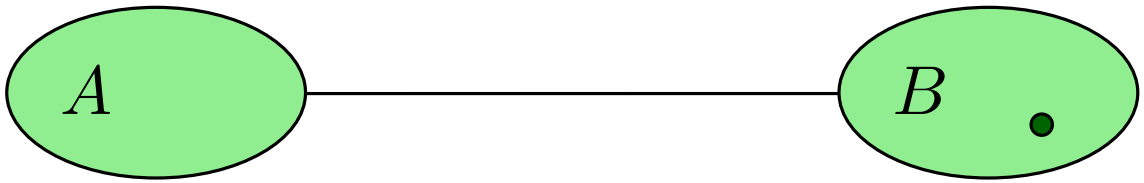


and the call graph:

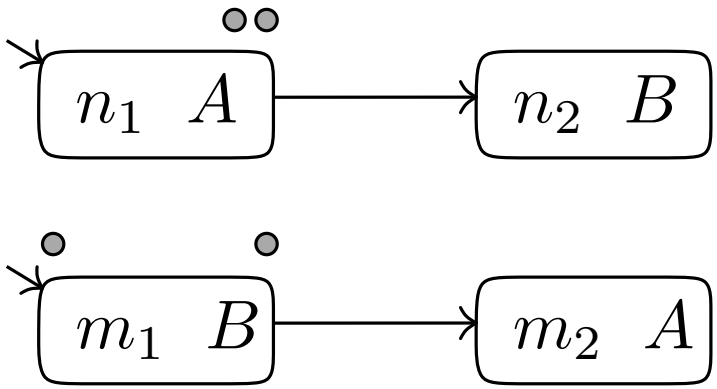


Example of Deadlock

Two sites, with two resources each:

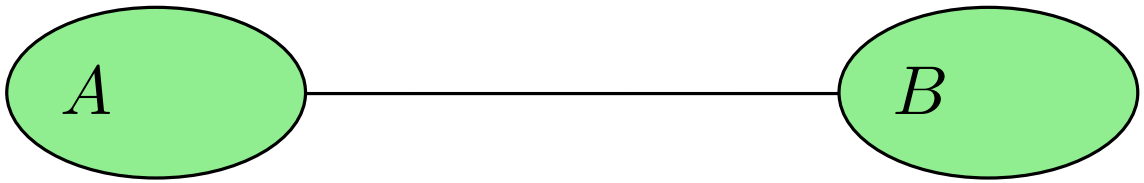


and the call graph:

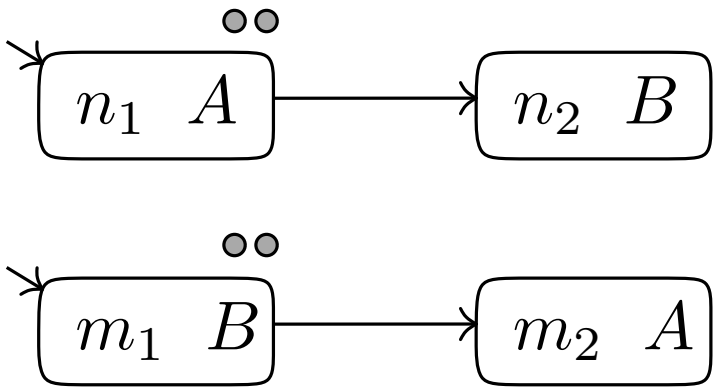


Example of Deadlock

Two sites, with two resources each:

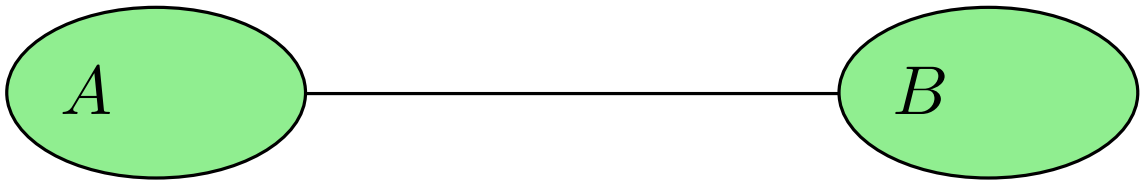


and the call graph:

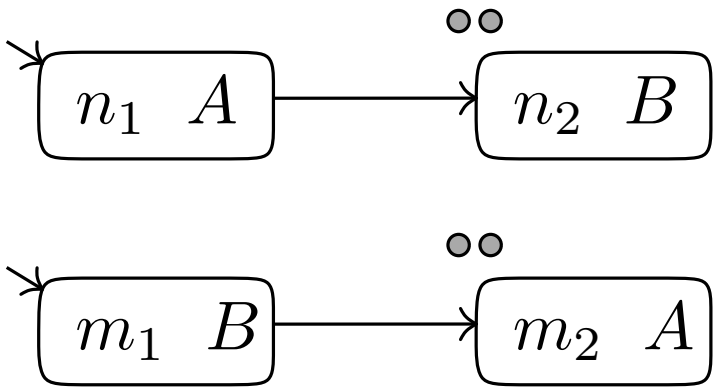


Example of Deadlock

Two sites, with two resources each:



and the call graph:



Summary of Contributions

Contribution #1

Efficient deadlock Avoidance can is possible provided call-graphs are know statically

Summary of Contributions

Contribution #1

Efficient deadlock Avoidance can is possible provided call-graphs are know statically

Contribution #2

Optimal annotations can be efficiently computed.
If annotations are not followed *anomalies* can occurr.

Summary of Contributions

Contribution #1

Efficient deadlock Avoidance can is possible provided call-graphs are know statically

Contribution #2

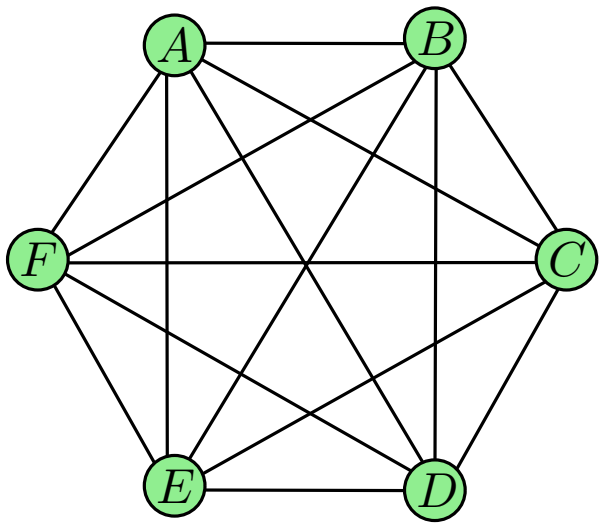
Optimal annotations can be efficiently computed.
If annotations are not followed *anomalies* can occurr.

Contribution #3

Distributed Deadlock Avoidance with (individual) liveness guarantees can be efficiently achieved.

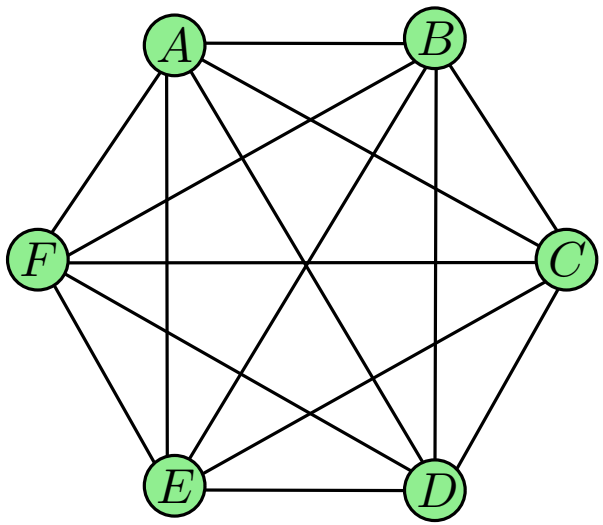
Model of Computation

- Remote procedure call (with Wait-On-Connection)
- Asynchronous messages
- All to all communication
- Finite resources: T_A total number of threads



Model of Computation

- Remote procedure call (with Wait-On-Connection)
- Asynchronous messages
- All to all communication
- Finite resources: T_A total number of threads



- We seek a *deadlock avoidance* solution with **no extra communication**

Distributed Deadlock Avoidance Solution

Two parts:

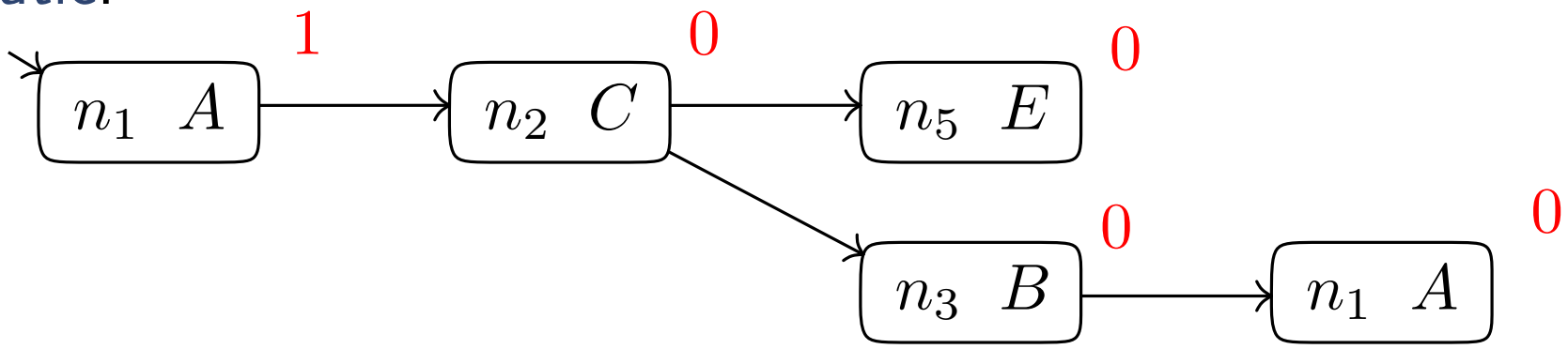
1. Static:

2. Dynamic:

Distributed Deadlock Avoidance Solution

Two parts:

1. Static:

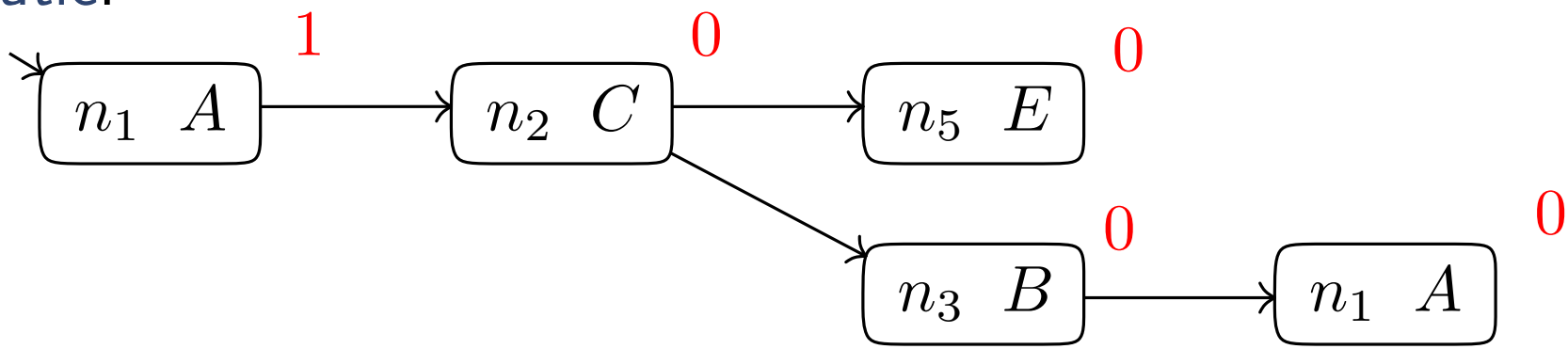


2. Dynamic:

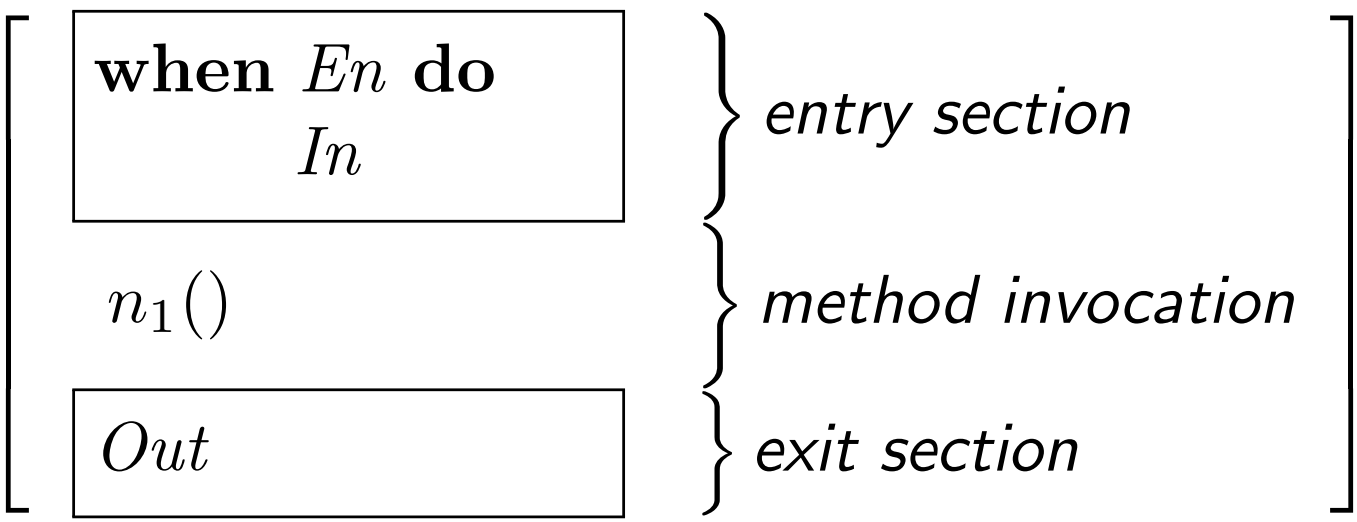
Distributed Deadlock Avoidance Solution

Two parts:

1. Static:

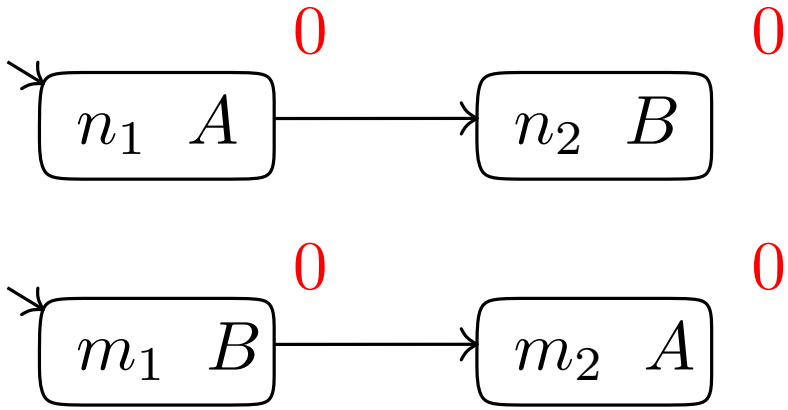


2. Dynamic:



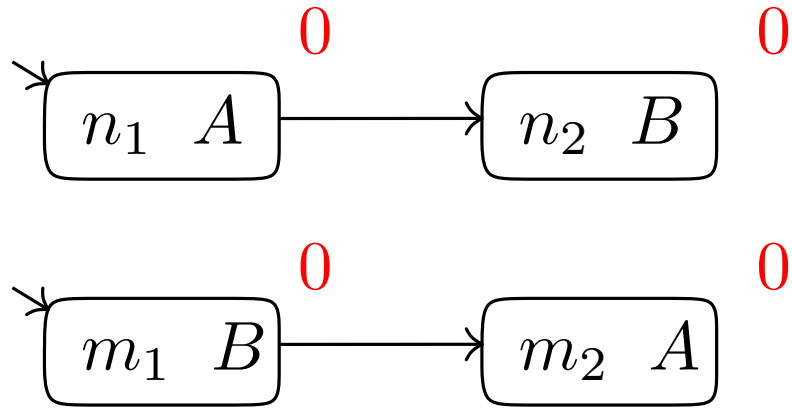
Annotations

Annotations are computed statically



Annotations

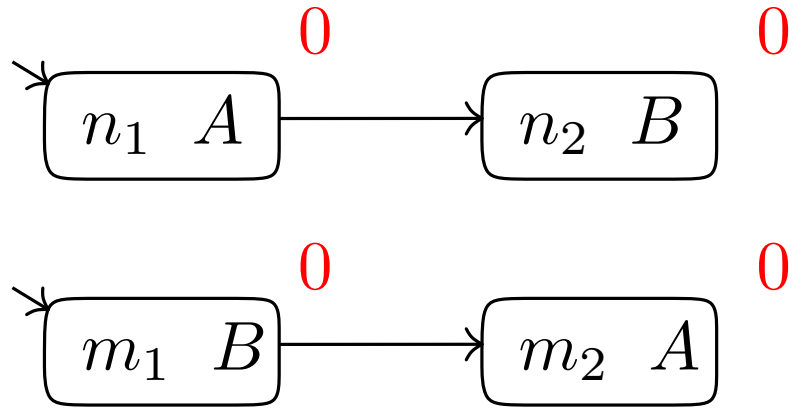
Annotations are computed statically



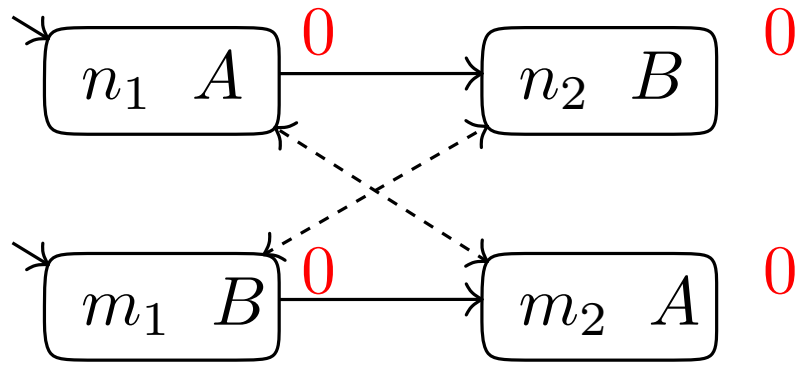
Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.

Annotations

Annotations are computed statically

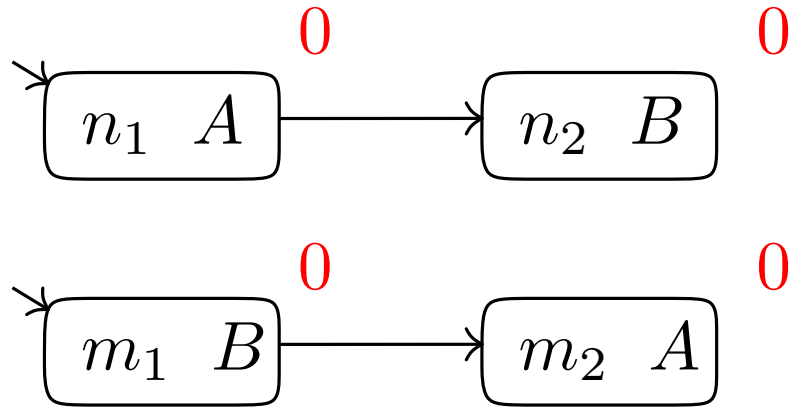


Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.

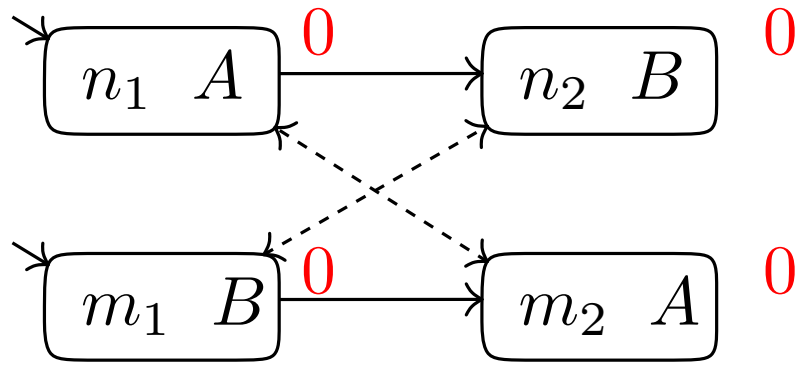


Annotations

Annotations are computed statically



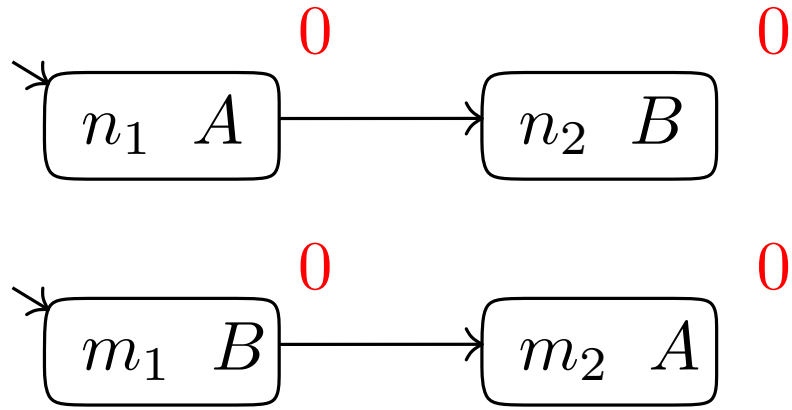
Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.



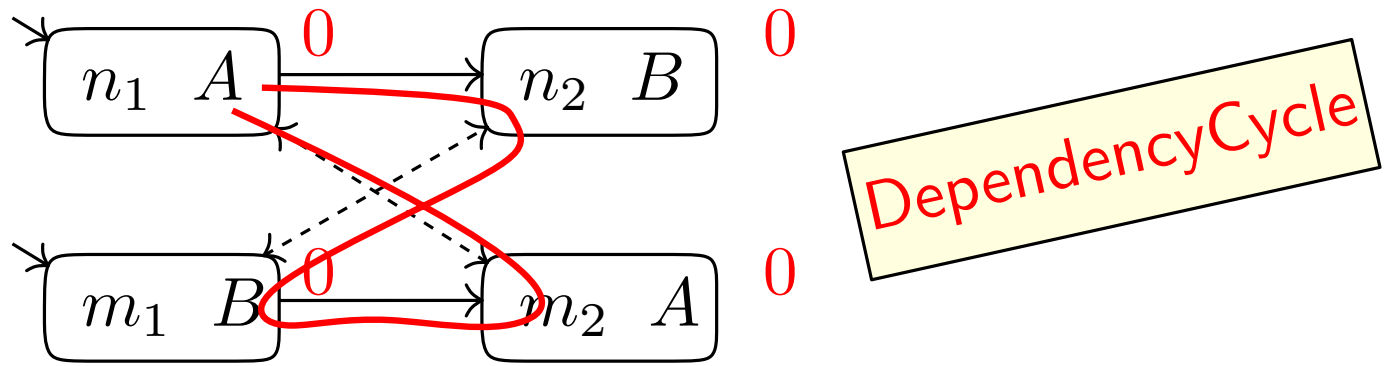
- n depends on m if there is a path from n to m containing a $\rightarrow$
- A dependency cycle is close path with a $\rightarrow$

Annotations

Annotations are computed statically



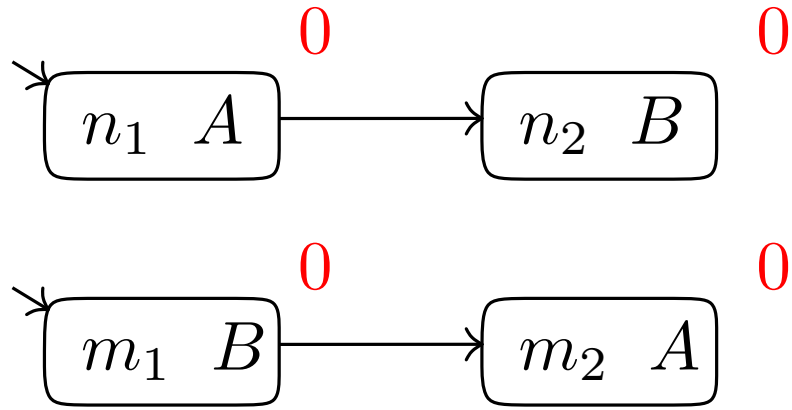
Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.



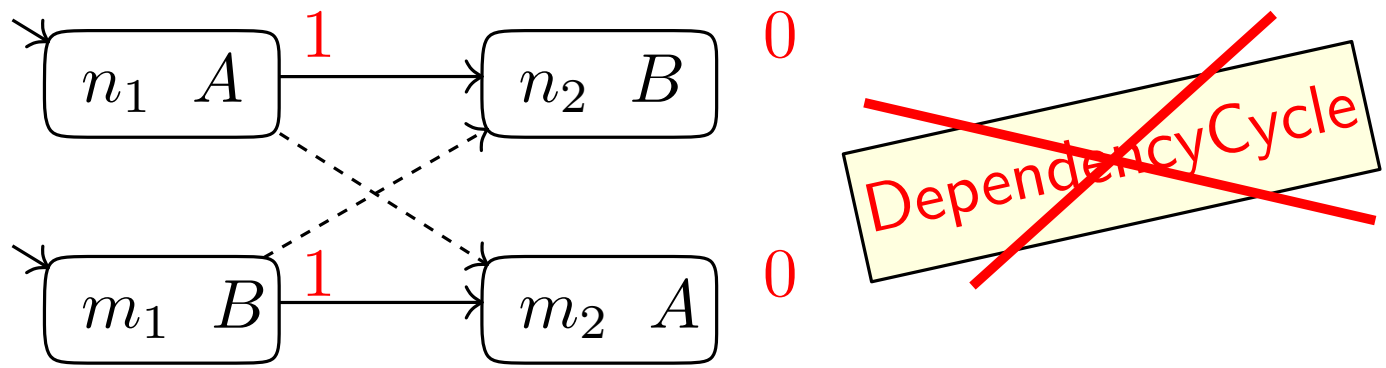
- n depends on m if there is a path from n to m containing a $\rightarrow$
- A dependency cycle is close path with a $\rightarrow$

Annotations

Annotations are computed statically



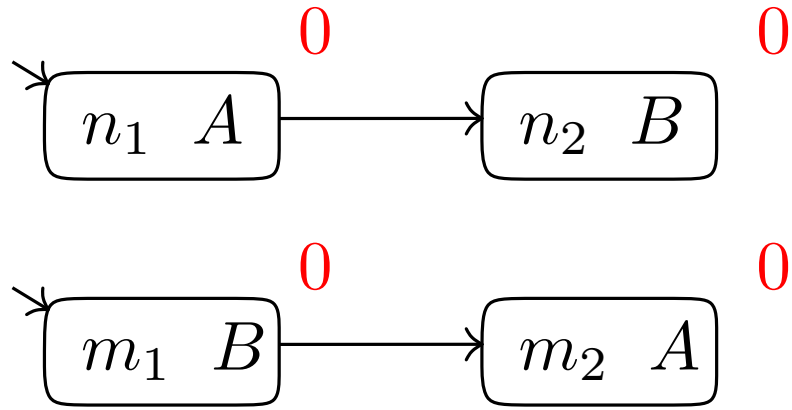
Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.



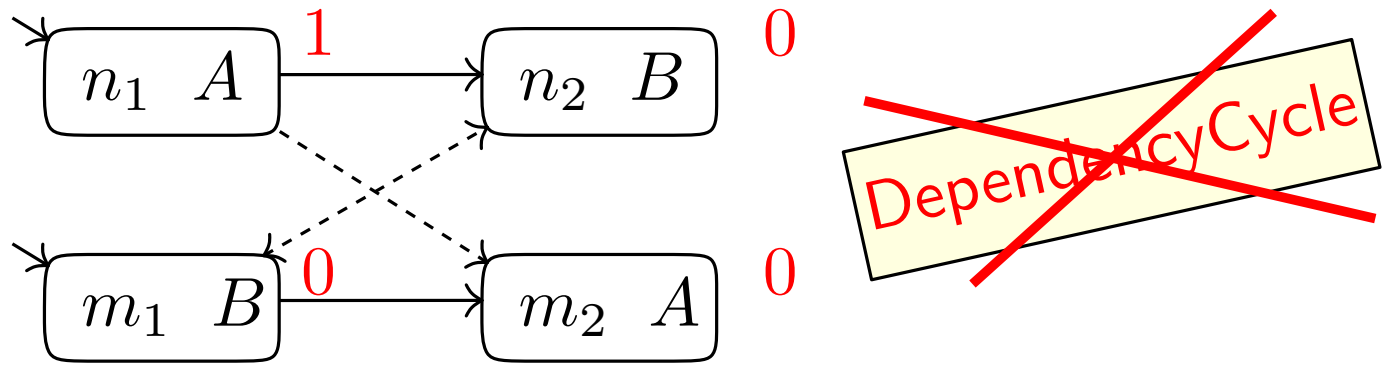
- n depends on m if there is a path from n to m containing a $\rightarrow$
- A dependency cycle is close path with a $\rightarrow$

Annotations

Annotations are computed statically



Dependency edges $n \dashrightarrow m$ whenever $\alpha(n) \leq m$ for two calls in the same node.



- n depends on m if there is a path from n to m containing a $\rightarrow$
- A dependency cycle is close path with a $\rightarrow$

Basic Solution Deadlock Avoidance

Protocol Basic-P:

$$\boxed{n_1 \;\; A}^{\alpha}$$

$$\left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } \alpha < t_A \textbf{ do} \\ t_A{-}{-}\end{array}\right] \\ n_1() \\ t_A{+}{+}\end{array}\right]$$

Basic Solution Deadlock Avoidance

Protocol Basic-P:

$$\boxed{n_1 \;\; A}^{\alpha}$$

$$\left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } \alpha < t_A \textbf{ do} \\ t_A{-}{-}\end{array}\right] \\ n_1() \\ t_A{+}{+}\end{array}\right]$$

Theorem: If α has no cyclic dependencies, then Basic-P guarantees absence of deadlock.

Basic Solution Deadlock Avoidance

Protocol Basic-P:

$$\boxed{n_1 \;\; A}^{\alpha}$$

$$\left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } \alpha < t_A \textbf{ do} \\ t_A{-}{-}\end{array}\right] \\ n_1() \\ t_A{+}{+}\end{array}\right]$$

Theorem: If α has no cyclic dependencies, then Basic-P guarantees absence of deadlock.

Lemma: The following is an **invariant**:

> *The number of processes running methods with annotation i or higher is at most $T_A - i$.*

The Annotation Theorem

Theorem: If α has no cyclic dependencies, then $\textsc{Basic-P}$ guarantees absence of deadlock.

Lemma: The following is an **invariant**:

> *The number of processes running methods with annotation i or higher is at most $T_A - i$.*

annotation α	Max num of procs
0	T_A
1	$T_A - 1$
$\ldots$	$\ldots$
$T_A - 1$	1

The Annotation Theorem

Theorem: If α has no cyclic dependencies, then BASIC-P guarantees absence of deadlock.

Lemma: The following is an **invariant**:

> *The number of processes running methods with annotation i or higher is at most $T_A - i$.*

Lemma: If a request $\overset{\alpha}{\boxed{n_1 \;\; A}}$ is disabled, then there is an active process running $\overset{\alpha_2}{\boxed{n_2 \;\; A}}$ with $\alpha_2 \leq \alpha$.

Annotations

Two immediate questions:

1. How to compute acyclic annotations

2. What if annotations are not acyclic?

Annotations

Two immediate questions:

1. How to compute acyclic annotations

– Visit nodes following some reverse topological order.
– When visiting n, compute the set of nodes S previously visited and reachable following $(\rightarrow \cup \dashrightarrow)^*$.
– Set $\alpha(n)$ to 1 plus the largest node in S that resides in the same site.

2. What if annotations are not acyclic?

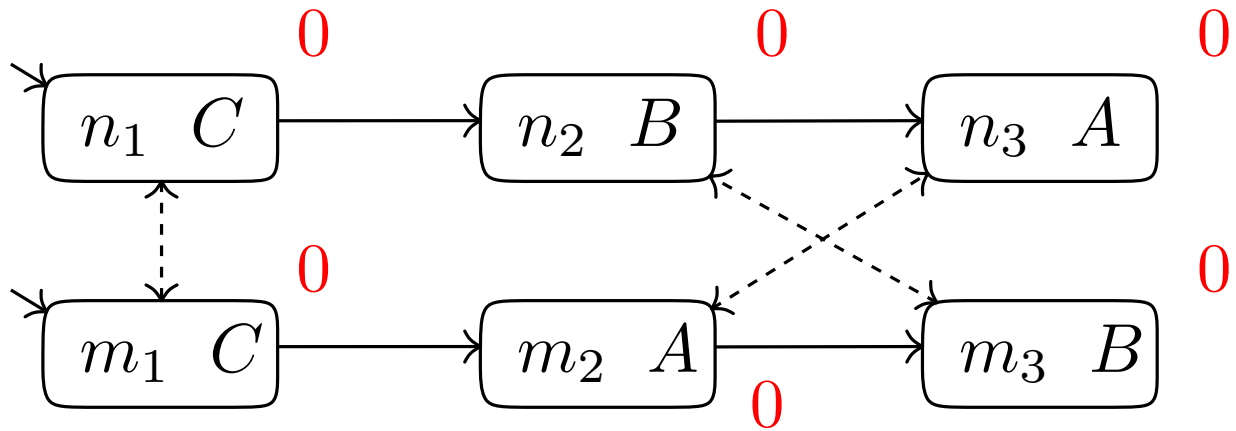
Annotations

Two immediate questions:

1. How to compute acyclic annotations

– Visit nodes following some reverse topological order.
– When visiting n, compute the set of nodes S previously visited and reachable following $(\rightarrow \cup \dashrightarrow)^*$.
– Set $\alpha(n)$ to 1 plus the largest node in S that resides in the same site.

2. What if annotations are not acyclic?



$T_A = 1$
$T_B = 1$
$T_C = 1$

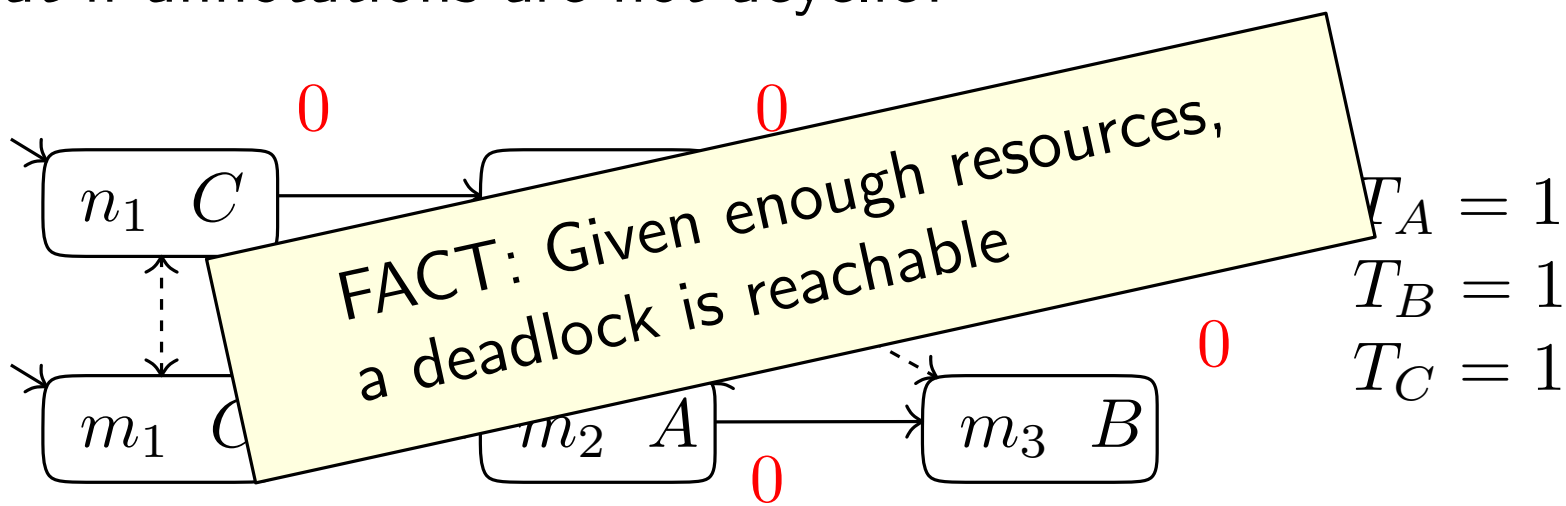
Annotations

Two immediate questions:

1. How to compute acyclic annotations

– Visit nodes following some reverse topological order.
– When visiting n, compute the set of nodes S previously visited and reachable following $(\rightarrow \cup \dashrightarrow)^*$.
– Set $\alpha(n)$ to 1 plus the largest node in S that resides in the same site.

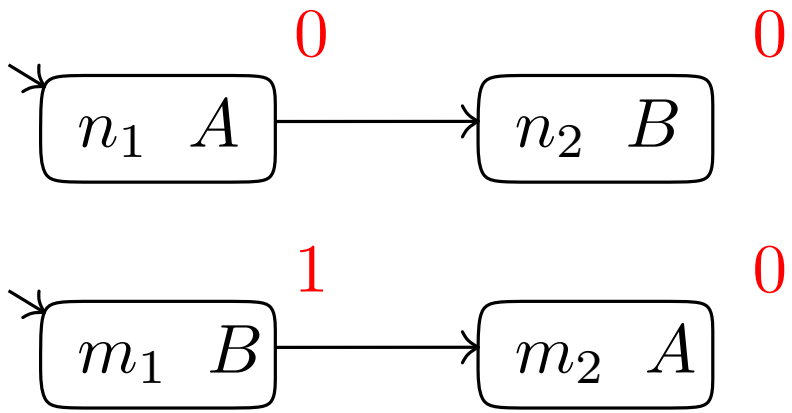
2. What if annotations are not acyclic?



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

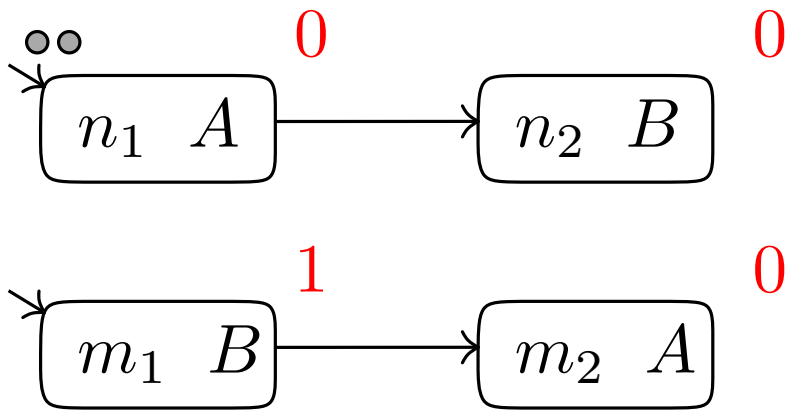
and the call graph:



How about liveness?

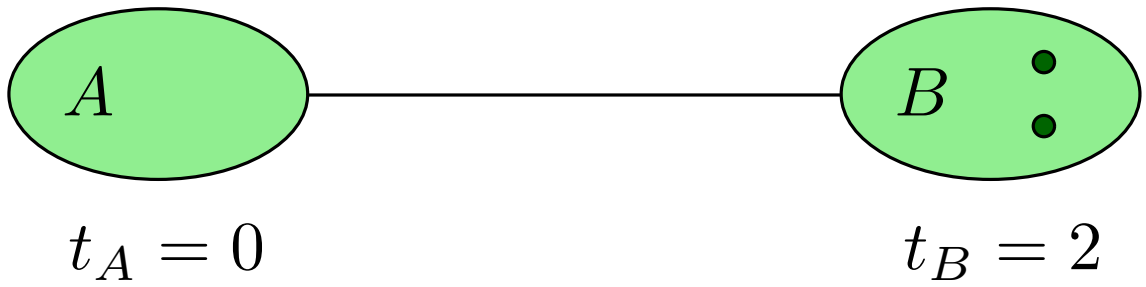
Consider two nodes, with two resource each ($T_A = T_B = 2$):

and the call graph:

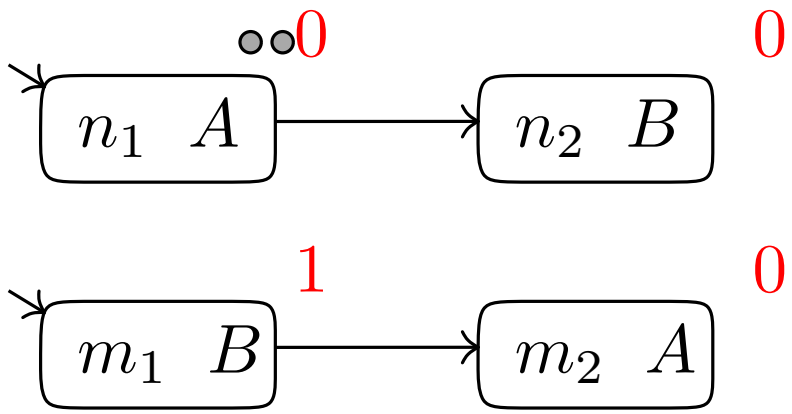


How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

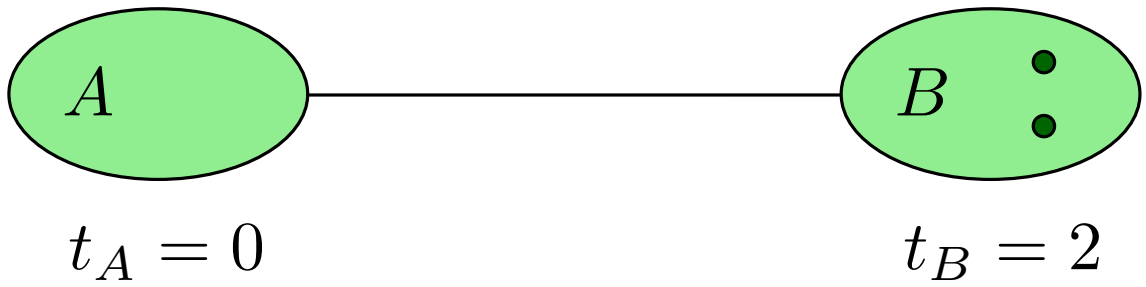


and the call graph:

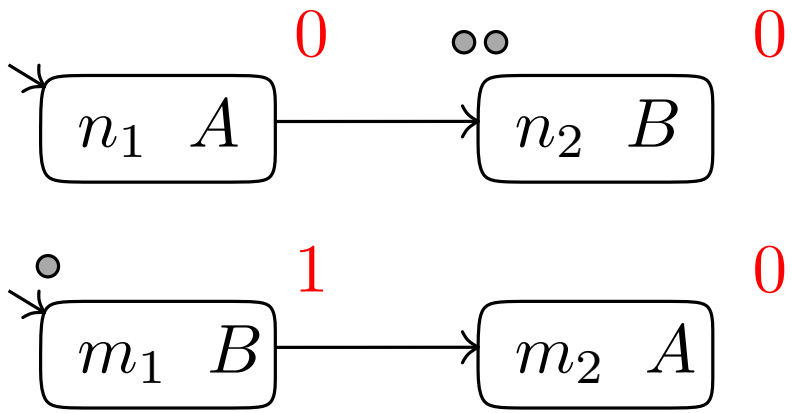


How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):



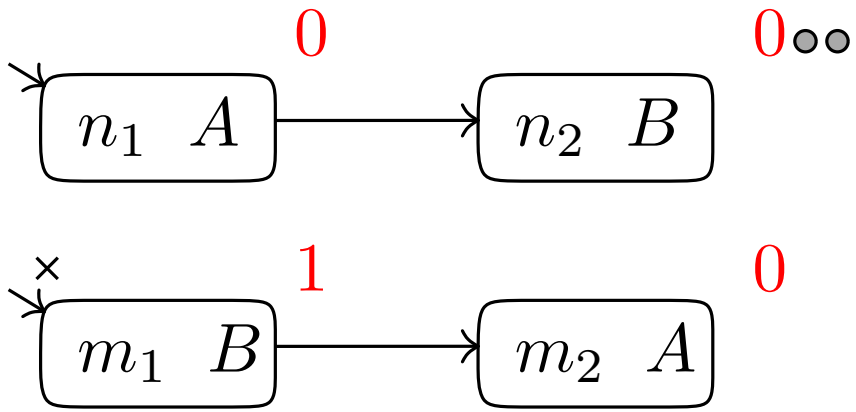
and the call graph:



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

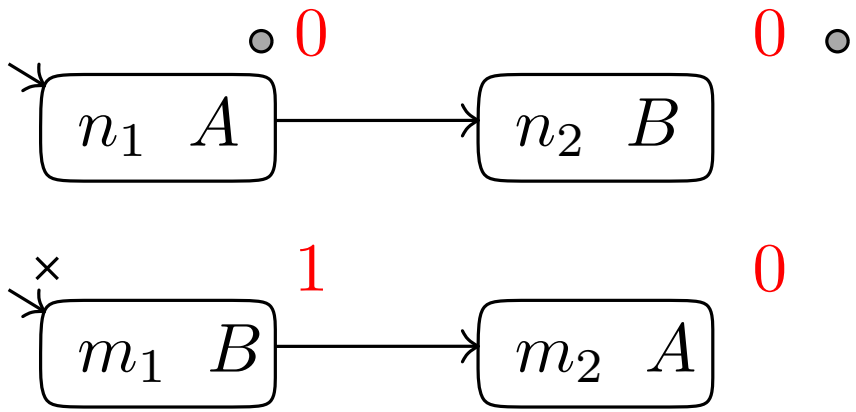
and the call graph:



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

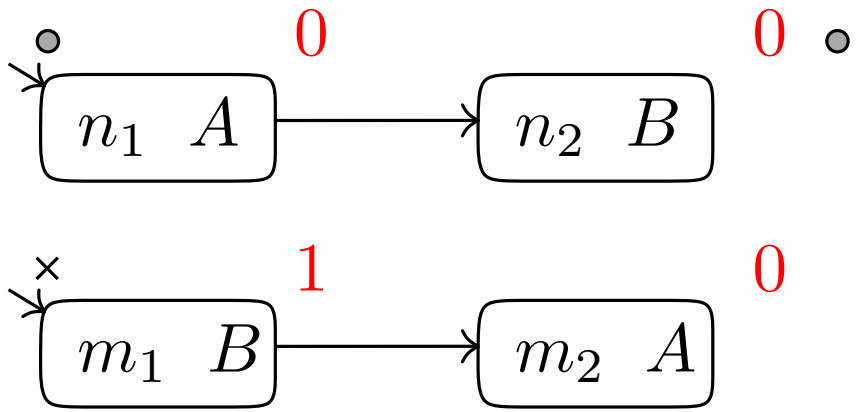
and the call graph:



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

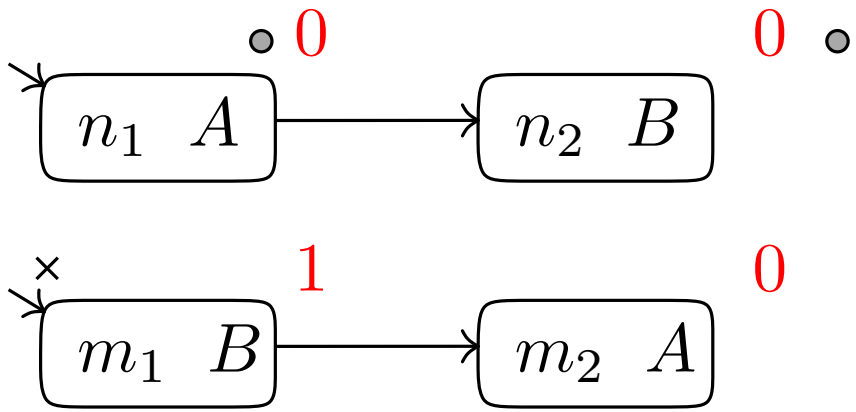
and the call graph:



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

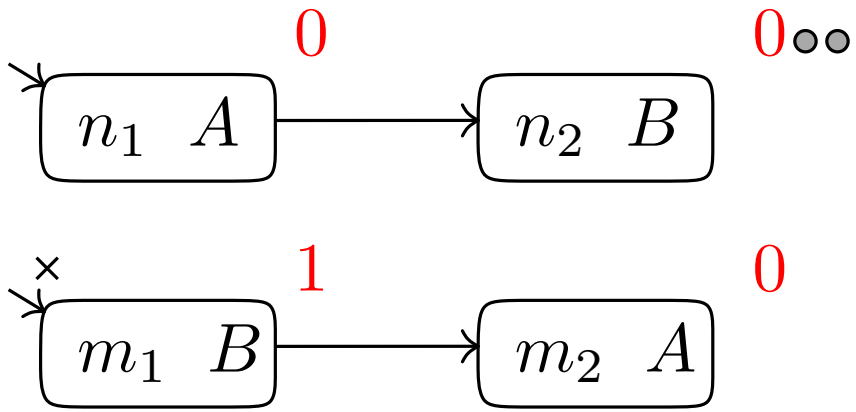
and the call graph:



How about liveness?

Consider two nodes, with two resource each ($T_A = T_B = 2$):

and the call graph:



Revisiting the Invariant

Lemma: The following is an **invariant**:

The number of processes running methods with annotation i or higher is at most $T_A - i$.

$$act_{A,\geq i} \leq T_A - i \qquad \text{for all notes } A \text{ and } i$$

where

$act_{A,i}$: number of active processes in A with annotation i

$act_{A,\geq i} = \sum_{k \geq i} act_{A,i}$

Revisiting the Invariant

Lemma: The following is an **invariant**:

The number of processes running methods with annotation i or higher is at most $T_A - i$.

$$act_{A,\geq i} \leq T_A - i \qquad \text{for all notes } A \text{ and } i$$

where

$act_{A,i}$: number of active processes in A with annotation i

$act_{A,\geq i} = \sum_{k \geq i} act_{A,i}$

The weakest precondition on allowing a request for $\boxed{n \;\; A}^{\,i}$

$$\varphi' = \bigwedge_k \begin{cases} act_{A,\geq k} \leq T_A - k & \text{if } k > i \\ act_{A,\geq k} + 1 \leq T_A - k & \text{if } k \leq i \end{cases}$$

The Protocol Live-P

To execute $\boxed{n\ A}^{i}$:

$$\left[\begin{array}{l} \left[\begin{array}{l}\textbf{when } \varphi' \textbf{ do}\\ \quad act_{A,i}{+}{+}\end{array}\right] \\ \\ n() \\ \\ act_{A,i}{-}{-} \end{array}\right]$$

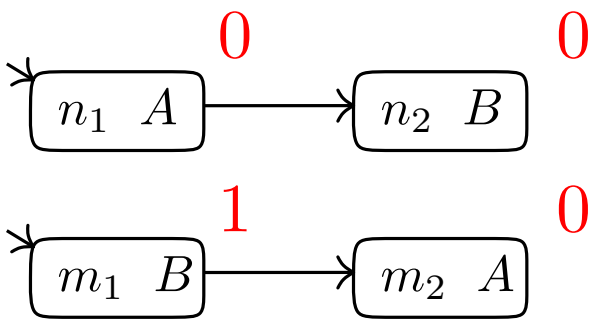
Theorem (Deadlock Avoidance): If α is acyclic, then $\textsc{Live-P}$ guarantees absence of deadlock.

Theorem (Liveness): If α is acyclic, then $\textsc{Live-P}$ guarantees that every waiting process is eventually enabled.

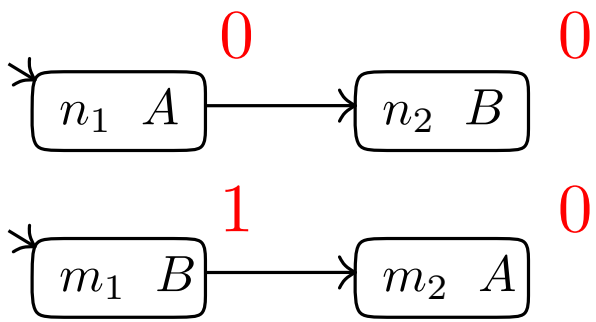
Live-P provides liveness

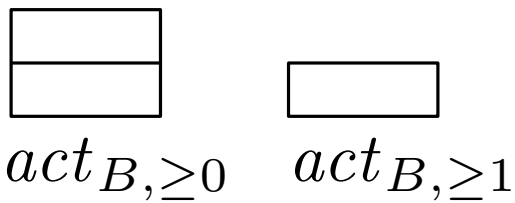
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P



BASIC-P

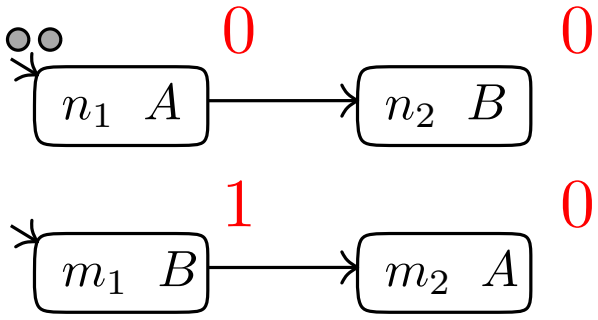


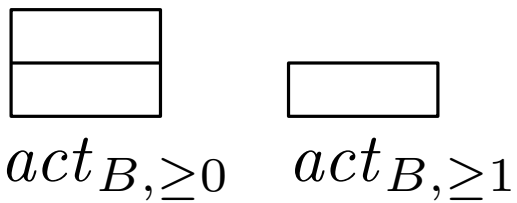


Live-P provides liveness

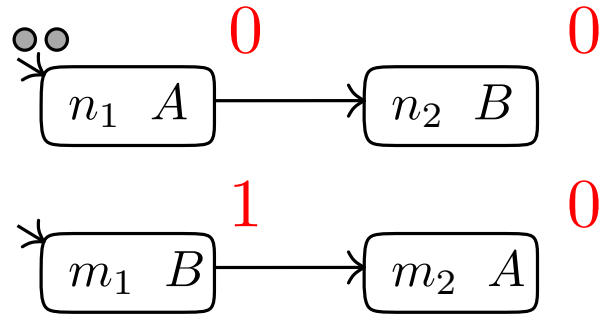
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P





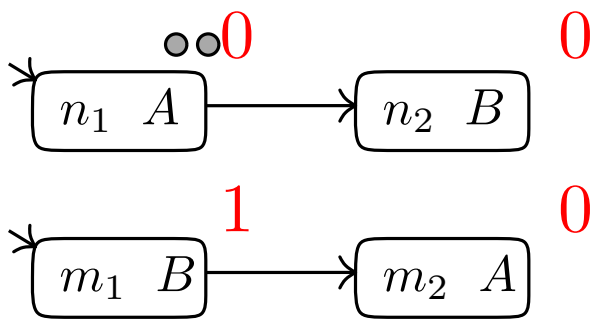
BASIC-P

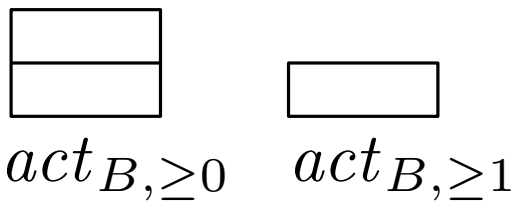


Live-P provides liveness

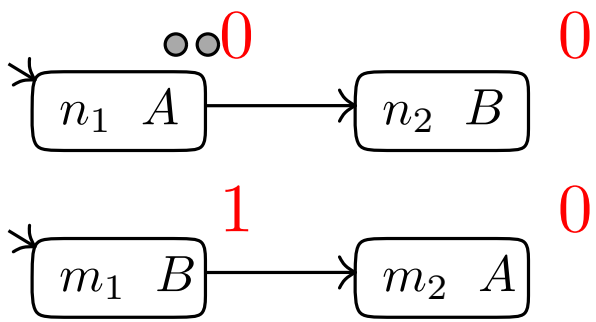
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P





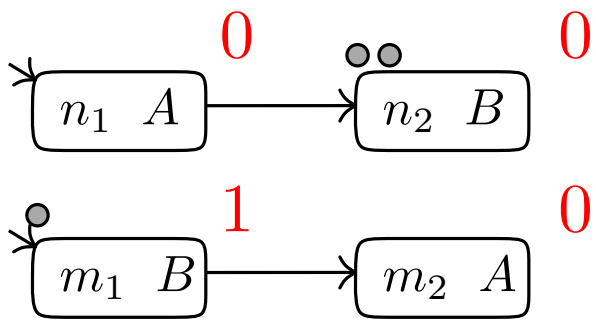
BASIC-P

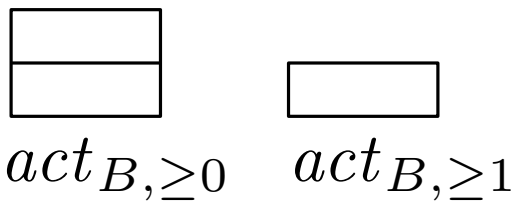


Live-P provides liveness

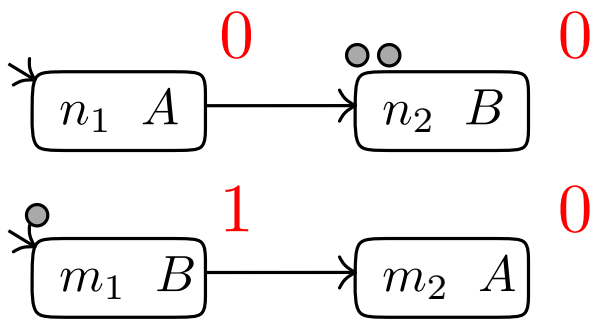
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P





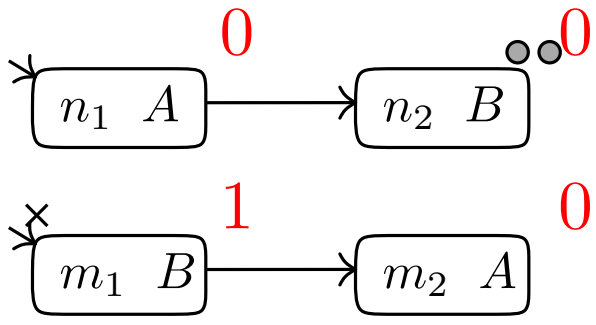
BASIC-P



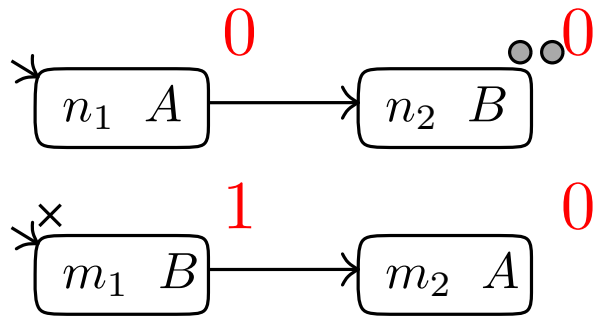
Live-P provides liveness

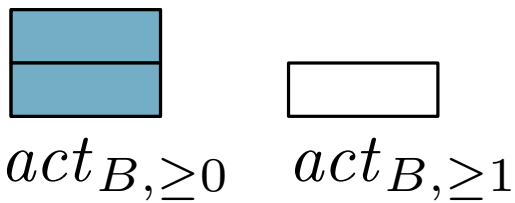
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P



BASIC-P

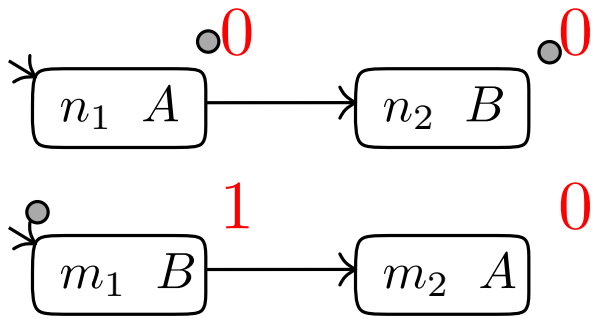




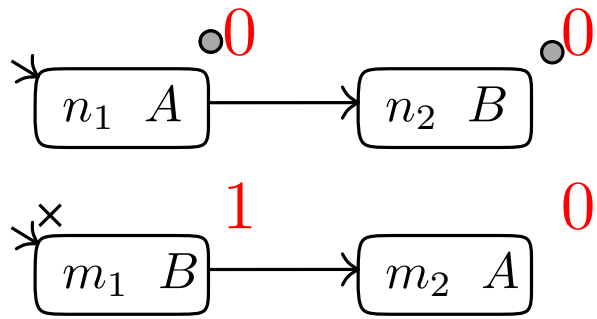
Live-P provides liveness

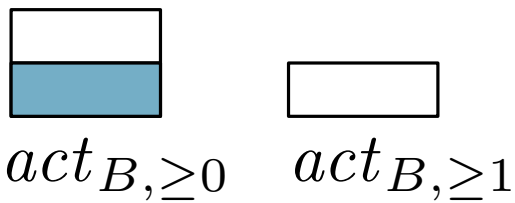
Consider two nodes, with two resource each ($T_A = T_B = 2$):

LIVE-P



BASIC-P





Conclusions

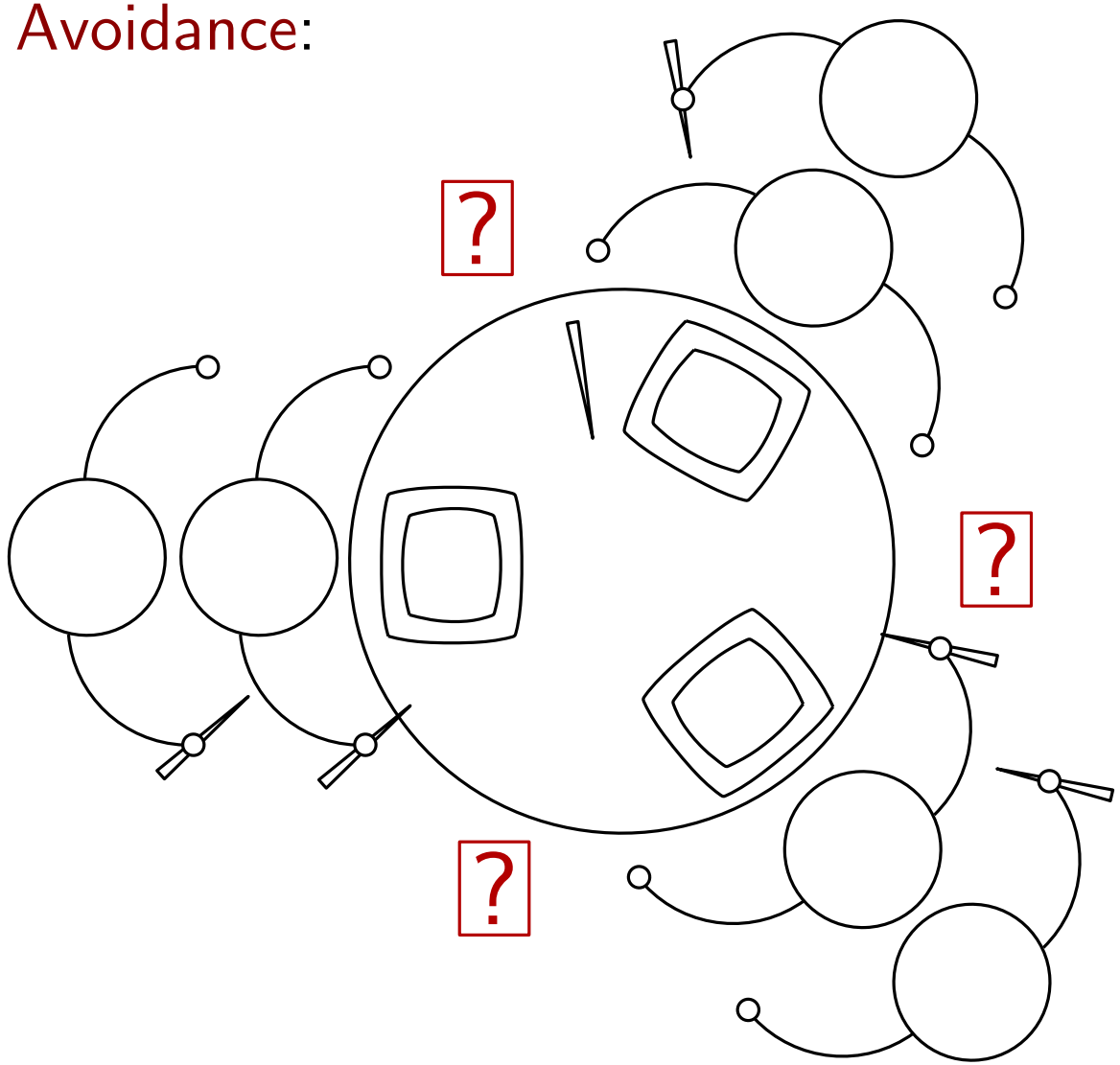
- Distributed Deadlock Avoidance is possible without communication
- . . . provided call-graphs are known
- Using static annotations + runtime protocols
- If cycles are allowed (e.g. by uncontrolled resource allocation), then deadlocks are unavoidable, provided enough resources
- Individual liveness is also enforceable
- Future work:
  - is deadlock avoidance enforceable for any amount of initial resources?
  - can this be adapted to composable conveyor systems?

Conclusions

- Distributed Deadlock Avoidance is possible without communication
- . . . provided call-graphs are known
- Using static annotations + runtime protocols
- If cycles are allowed (e),
  then deadlocks are una
- Individual liveness is al
- Future work:
  - is deadlock avoidan l resources?
  - can this be adapted to composable conveyor systems?

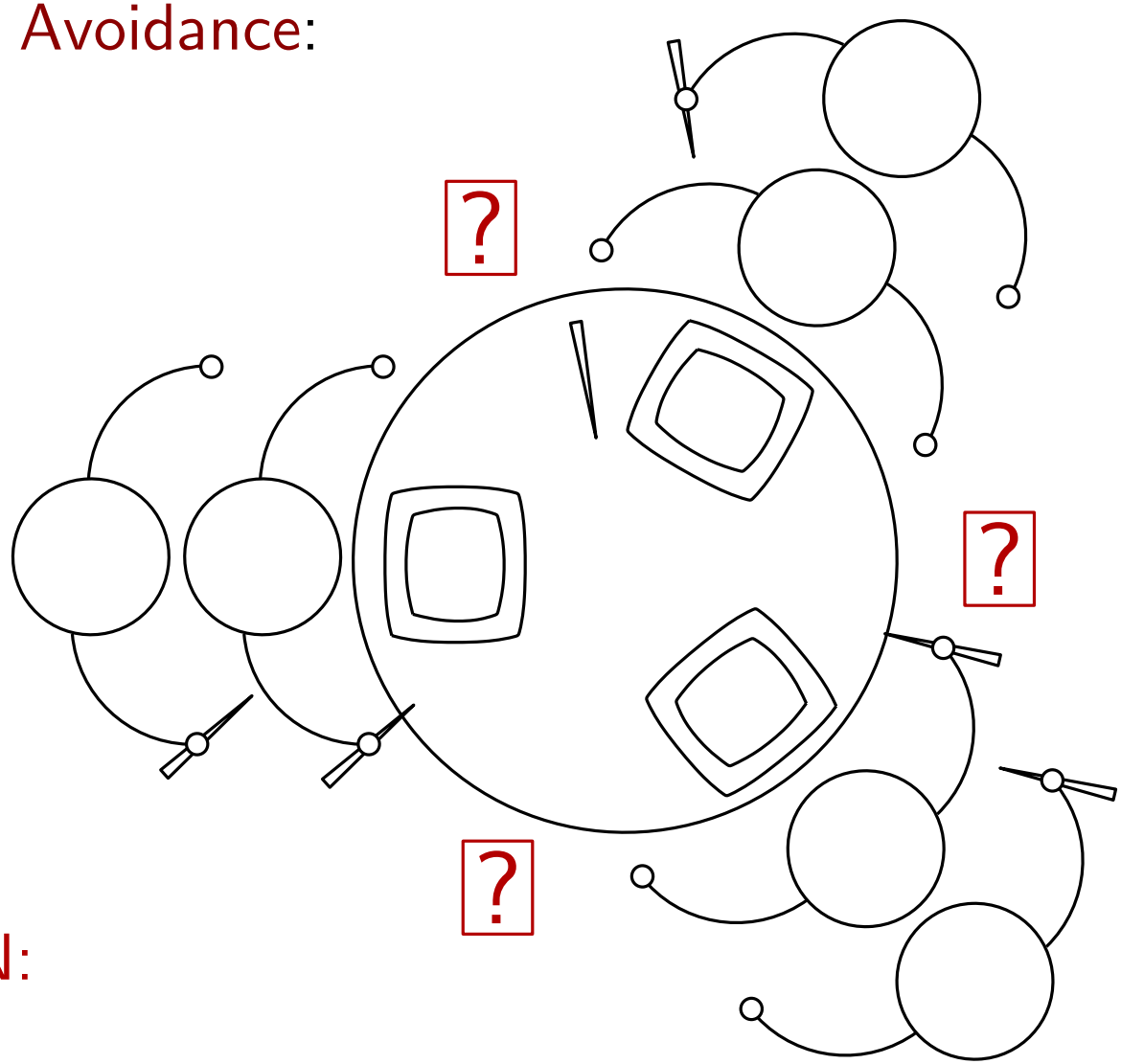
Distributed Dinning Philosphers

Distributed Avoidance:



Distributed Dinning Philosphers

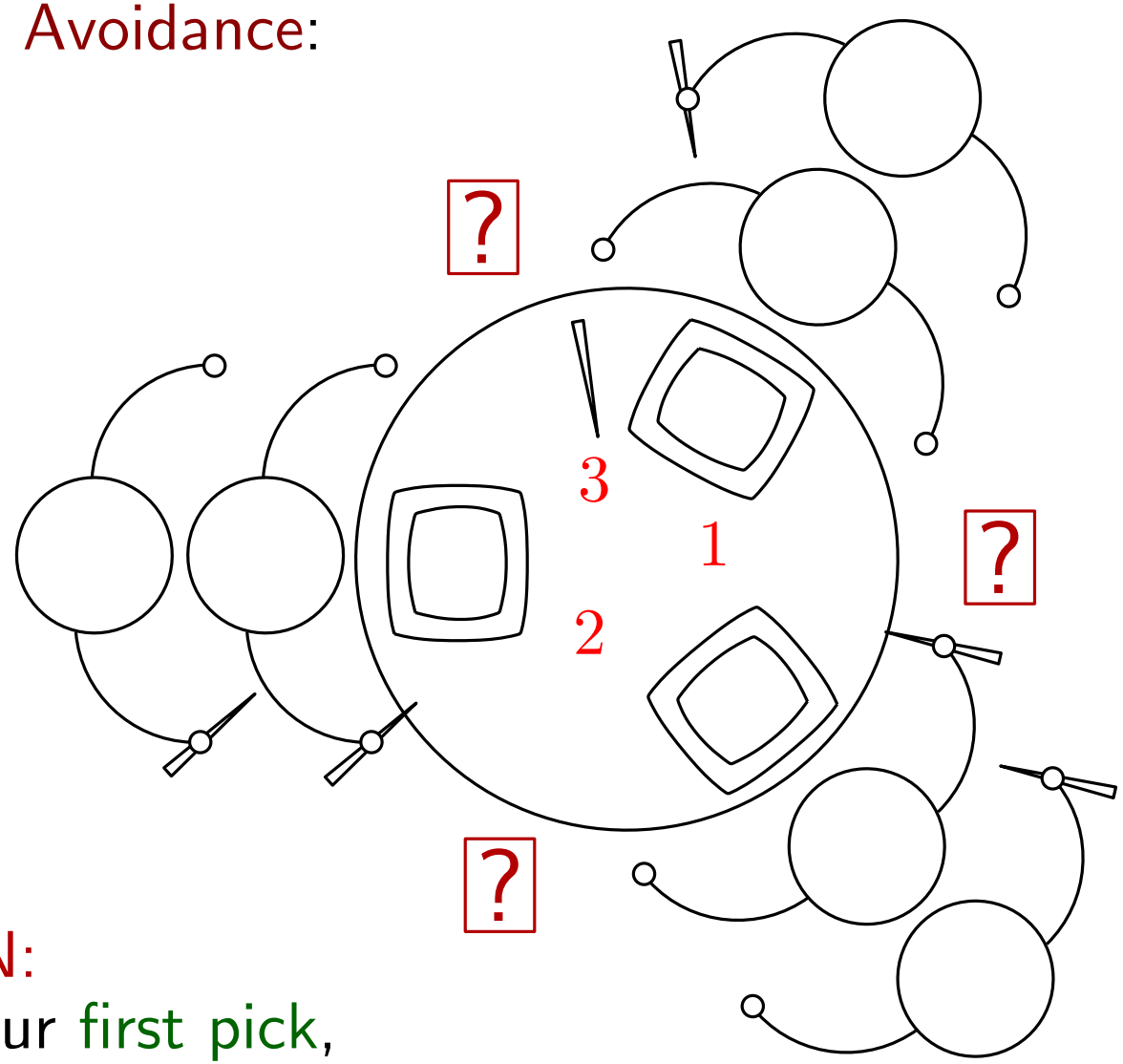
Distributed Avoidance:



SOLUTION:

Distributed Dinning Philosphers

Distributed Avoidance:



SOLUTION:

For your first pick,

do not the take last fork if going in increasing order.

For your second pick,

do as you wish.

Questions

Thank you for your attention!