Profile-Guided Java Program Partitioning for Power Aware Computing

Sriraman Tallam

and

Rajiv Gupta

Computer Science Department, The University of Arizona

Handhelds are powerful but power constrained !!

- **Mobile Devices are very powerful.**
  - The iPAQ handheld
    - 400 MHz. Intel Xscale Processor
    - 128 MB RAM
    - Wireless communication capabilities
- **Battery Power is a bottleneck.**
  - Processing and Communication hog the battery resource, drive handheld at peak power.
  - No-op just needs idle power.
  - Inability to run power hungry applications.

Remote Wireless Computing scenario

- **Proliferation of Wireless access points.**
  - **Wi-Fi** Hotspots are now available widely.
- **Scenario**
  - The handheld can connect wirelessly to remote servers and use their computing cycles.
  - Opportunity to partly compute remotely and save power.
  - Relevant program parts need to be transmitted – DRAWBACK !!
  - Look at overall energy savings.

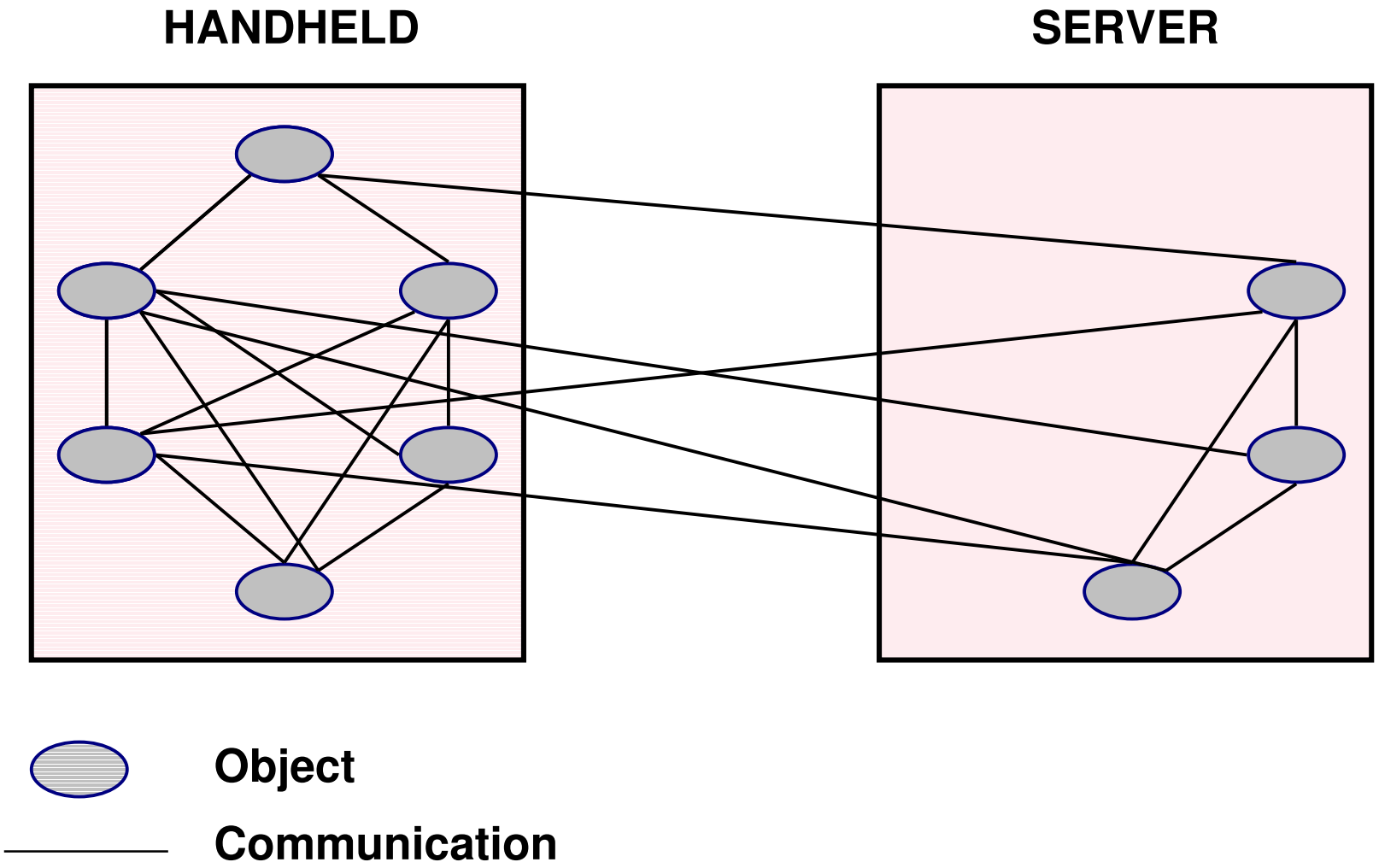
Problem

- **Is it possible to execute Java applications remotely and save power ?**
- **Execute Java applications partly remotely and reduce overall power.**
  - Remote computation runs CPU at idle power.
  - Transmission needs additional energy.
- **We form a partition of the Java program.**
  - Distribute objects between local and remote machines.
  - Local objects need peak CPU power.
  - Remote objects need idle CPU power.
  - Remote object creation needs transmission.

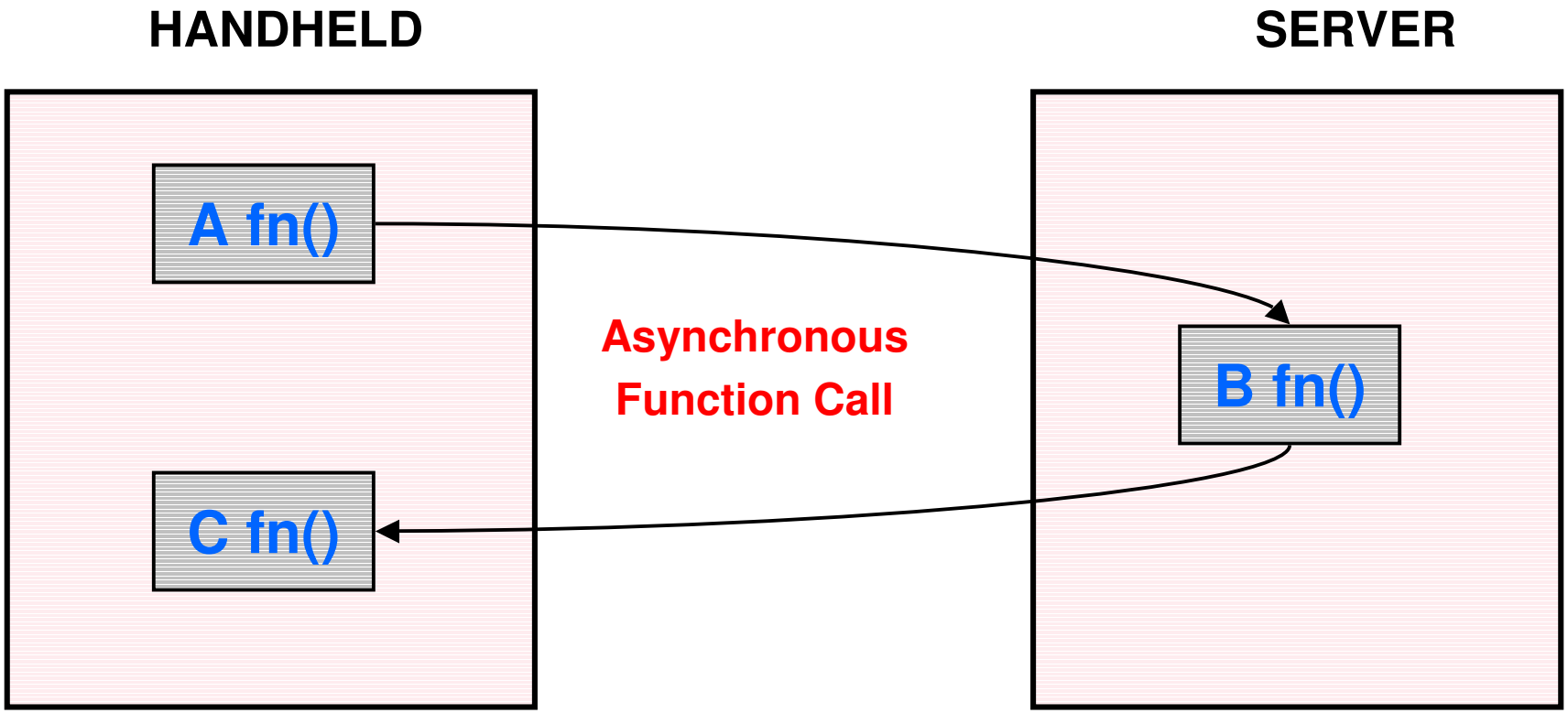
Overview of the Talk

- **Problem Statement** ✓
- **What is a Java program partition ?**
- **Partitioning Algorithm I - OPTIMAL**
  - Based on MIN-CUT and OPTIMAL
- **Drawbacks of Algorithm I**
- **Partitioning Algorithm II - FAST**
  - Greedy Algorithm and APPROXIMATE.
- **Experimental Results**

What is a Java Program Partition ?



Executing a Partitioned Java Application



Partitioning Algorithms Overview

- **Profiling Step**
  - Number of objects of each class created.
  - The amount of communication involved between object pairs.
  - The CPU time spent in the context of each object.
- **Partitioning Step – Graph Representation**
  - OPTIMAL Method
    - Gives optimal partitions, very poor running time.
  - FAST Method
    - Does not give the best partitions but is very fast.
- **Executing the Application as per the Partition**

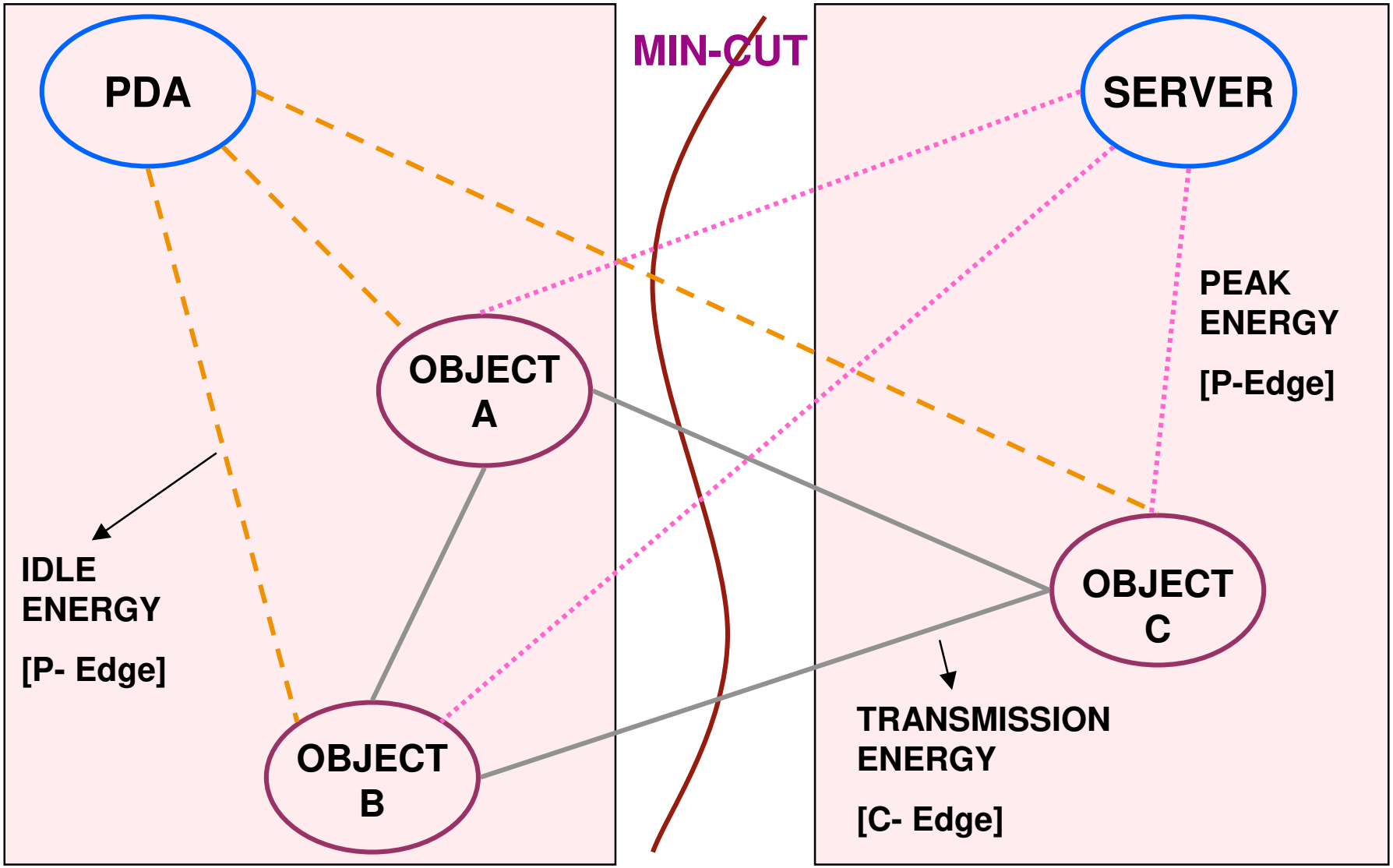
Profiling Step

- **The number of objects of each class created.**
- **The amount of communication involved between object pairs.**
  - This gives an estimate of the transmission energy needed if these object pairs were separated.

    Energy Spent= Transmission Energy per byte **X** Bytes Transmitted
- **The CPU time spent in the context of each object.**
  - This gives an estimate of the CPU energy saved if this object was executed remotely.

    Energy Saved= (Peak - Idle) **X** CPU Time taken

Partitioning Algorithm I
OPTIMAL



Using the ratio at run-time

- The partition outputs a ratio a : b of objects of every class.
- For every ‘a’ objects that are placed in the server, we place ‘b’ objects in the client.
- This way, a different execution instance can still use the partition.

OPTIMAL Partitioning

- **Graph Representation**
  - Each node is an object.
  - Edges represent communication [C-Edge] and computation energies [P-Edge].
- **Partitioning Algorithm**
  - MIN-CUT of this graph will give the OPTIMAL partition – expressed as a ratio.
- **Time and Space Complexity**
  - SPACE : Graph is as large as the number of objects created in the application, HUGE !!
  - TIME : runs in time cubic in the number of objects created, HUGE !!

Need a Faster Algorithm

- **Partitioning is dependent on network variables and handheld variables.**
  - Bandwidth.
  - Communication Energy per byte.
  - Battery Power.
- **Change of variables requires partitioning to be done again.**
- **TIME – SPACE inefficient algorithm might in itself need a lot of energy.**

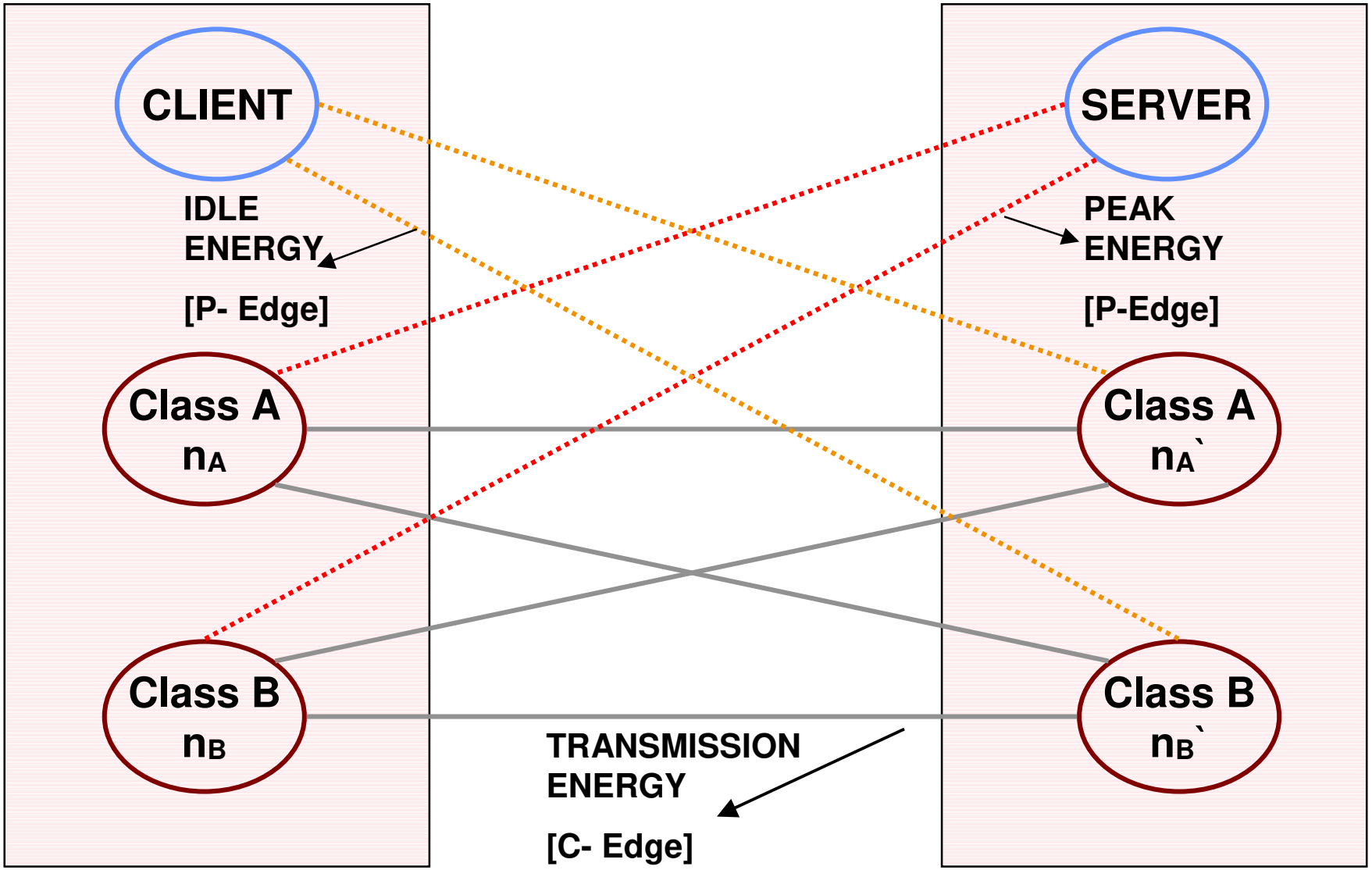
So Far

- **Problem Statement** ✓
- **What is a partition ?** ✓
- **Partitioning Algorithm I - OPTIMAL** ✓
  - Based on MIN-CUT and OPTIMAL
- **Drawbacks of Algorithm I** ✓
- **Partitioning Algorithm II - FAST**
  - Greedy Algorithm and APPROXIMATE.
- **Experimental Results**

Partition Algorithm II
FAST

- **Graph Representation – Minimize space**
  - Each node is a class.
  - Node weight is the number of objects of that class.
  - Edges represent communication [C-Edge] and computation energies [P-Edge], as before.
- **Partitioning Algorithm – Minimize time**
  - A greedy style algorithm gives a partition that tends to reduce the energy spent.

Partition Algorithm II



FAST Partitioning Algorithm

- **Benefit in moving an object to the server.**

  Benefit = (Peak Energy – Idle Energy)

- **Cost in moving an object to the server.**

  Cost = Transmission Energy Spent

  Cost = $((n_A - 1) * O_{AA} + n_B * O_{AB}) * \alpha$,

  O_{AA} is the average bytes transferred between objects of Class A.

- **If Benefit > Cost, the object can be moved to the server side.**

FAST Partitioning Algorithm

1. Compute benefit [v] and cost [w] for all objects.
2. *While* (there exists some object whose v/w > 0) *do*
3. Choose the class whose object has the maximum v/w ratio.
4. *if* maximum v/w > 1 *then*
5. Move all objects of this class.
6. recompute v/w for all objects, CHECKPOINT.

else

repeat

7. move one object to the server.
8. recompute v/w for all objects.
9. Find maximum v/w
10. *until* v/w >1

end-if.

end-while

Partition Algorithm II
FAST

- **TIME Complexity**
  - runs in time linear in the number of objects created in the worst-case.
- **SPACE Complexity**
  - Number of node and edges in the graph is a function of the number of classes, SMALL !!.

MATRIX Application

- **Java Application that finds a sub-matrix within a Large Matrix**
  - Image Processing involves similar operations.

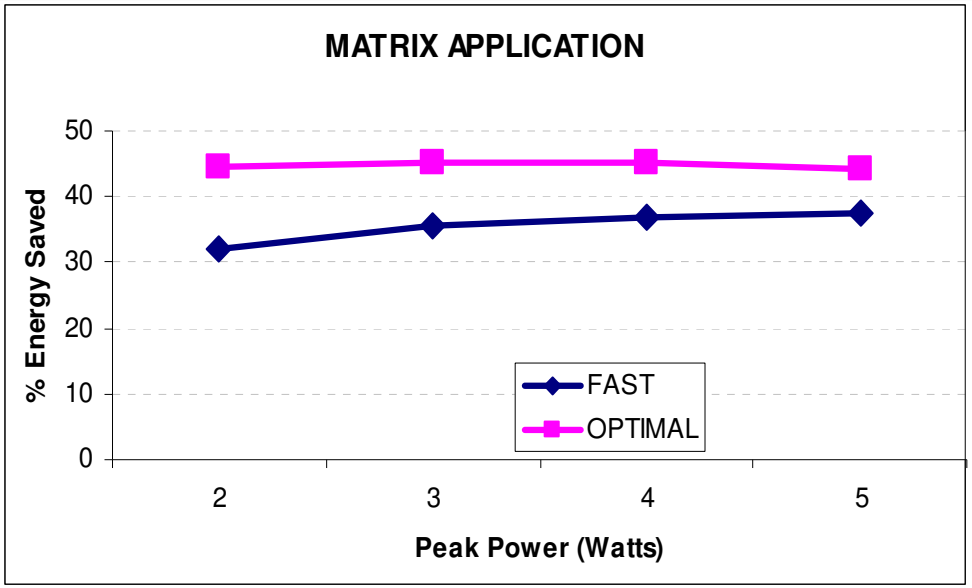
Number of Objects	Time Taken in secs.		Energy Savings %	
	OPTIMAL	FAST	OPTIMAL	FAST
3111	16.7	0.188	43.0	29.29
3111	16.7	0.188	44.08	29.29
3111	18.3	0.172	44.6	29.29
4443	95.1	0.133	43.6	32.22
6663	324	0.344	44.5	33.88

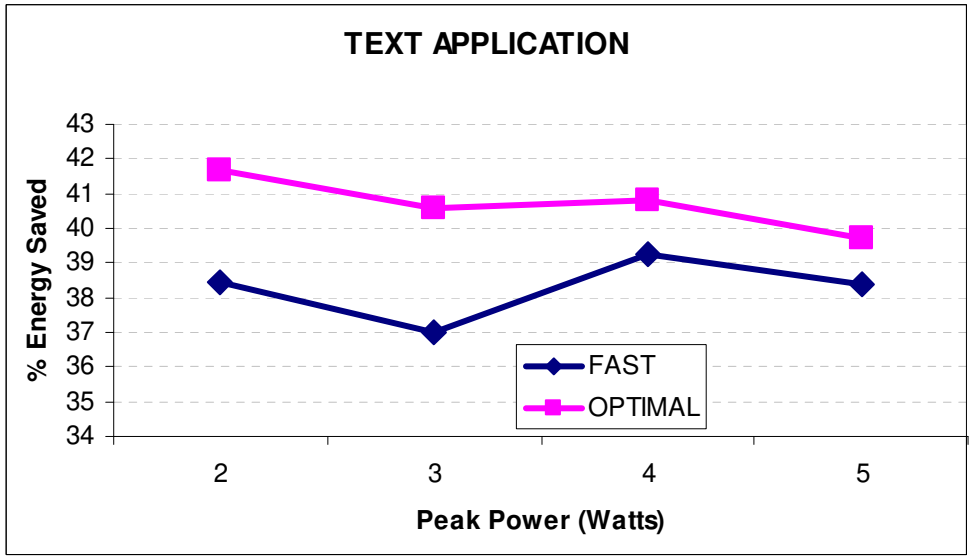
TEXT Application

- **Java Application that finds strings within large text documents**
  - Important operation in word-processing applications.

Number of Objects	Time Taken in secs.		Energy Savings %	
	OPTIMAL	FAST	OPTIMAL	FAST
523	1.62	0.003	45.29	37.84
1043	2.03	0.003	44.25	40.8
2043	8.94	0.06	42.1	38.06
3063	36.58	0.128	46.02	38.53
4563	41	0.172	47.5	43.06

Graphs for Energy Savings





Conclusions

- Energy savings in embedded devices by running Java programs partly remotely.
- We have presented an optimal algorithm that generates good partitions.
- We have presented a fast algorithm that can partition quickly.
- Experiments show that the fast algorithm is reasonably good in practice.