Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping

Georgios Goumas, Aristidis Sotiropoulos and Nectarios Koziris

National Technical University of Athens, Greece
Department of Electrical and Computer Engineering
Division of Computer Science

Computing Systems Lab

www.cslab.ece.ntua.gr
nkoziris@cslab.ece.ntua.gr

IPDPS 2001-San Francisco

Overview

Minimizing overall execution time of nested loops on multiprocessor architectures using message passing

How?

Loop Tiling for parallelism

+

Overlapping otherwise *interleaved* *communication* and pure *computation* sub-phases

Is it possible?

s/w communication layer + hardware should assist

OVERALL SCHEDULE IS LIKE A **PIPELINED DATAPATH**!

What is tiling or supernode transformation?

- Loop transformation
- Partitioning of iteration space J^n into n-D parallelepiped areas formed by n families of hyperplanes
- Each tile or supernode contains many iteration points within its boundary area
- Tile is defined by a square matrix H, each row vector h_i perpendicular to a family of hyperplanes
- Dually, tile is defined by n column vectors p_i which are its sides, P=[p_i]

It holds $P = H^{-1}$

Multilevel Tiling:
Tiling at all levels of memory hierarchy!

- ✓to increase reuse of register files
- ✓to increase reuse of cache lines (tiling for locality)
- ✓To increase locality in Virtual Memory

and at the **upper** level:

- ✓Tiling to exploit parallelism !

Why using tiling for parallelism?

- Increases Grain of Computation –

  Reduces synchronization points (atomic tile execution)

- Reduces overall communication cost (increases intraprocessor communication)

TRY TO FULLY UTILIZE ALL PROCESSORS

(CPUs !!!)

Tiling Transformation

Tiles are atomic, identical, bounded and sweep the index space

$$r : Z^n \rightarrow Z^{2n}, r(j) = \begin{bmatrix} \lfloor Hj \rfloor \\ j - H^{-1}\lfloor Hj \rfloor \end{bmatrix}$$

$\lfloor Hj \rfloor$ identifies the coordinates of the tile that j is mapped to

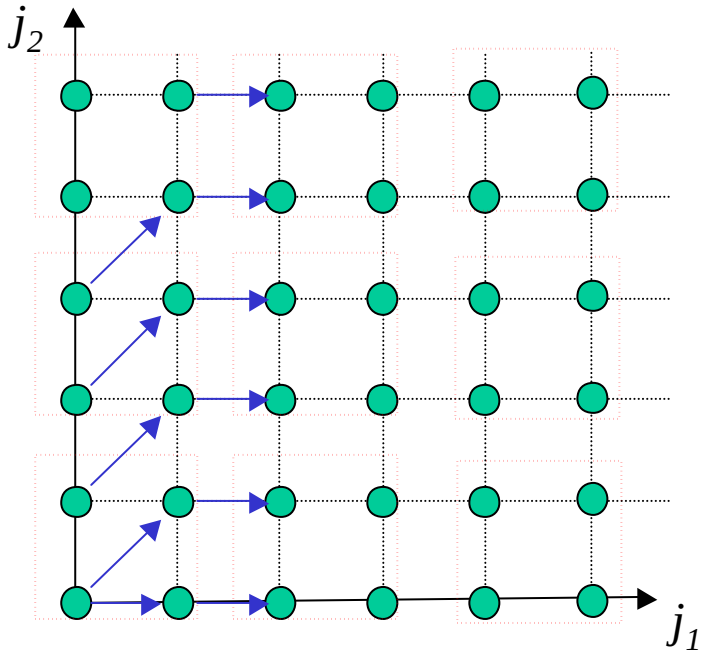
$j - H^{-1}\lfloor Hj \rfloor$ gives the coordinates of j within that tile relative to the tile origin

Example: A simple 2-D Tiling

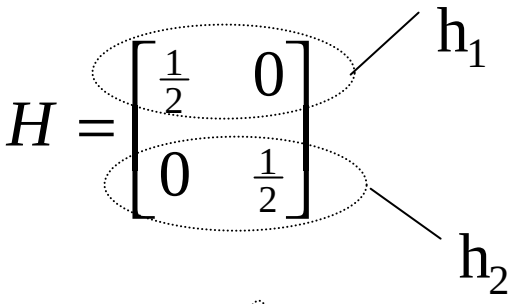
for $j_1 = 0$ to 5

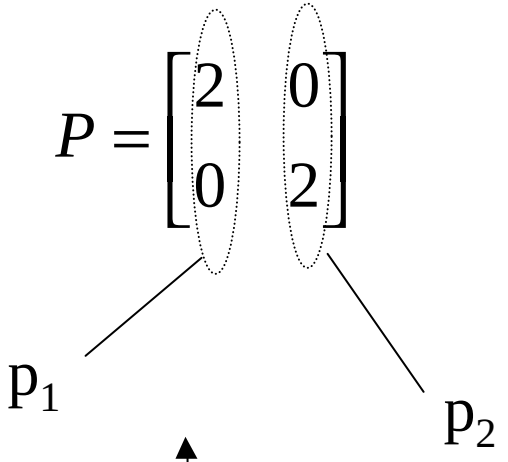
  for $j_2 = 0$ to 5

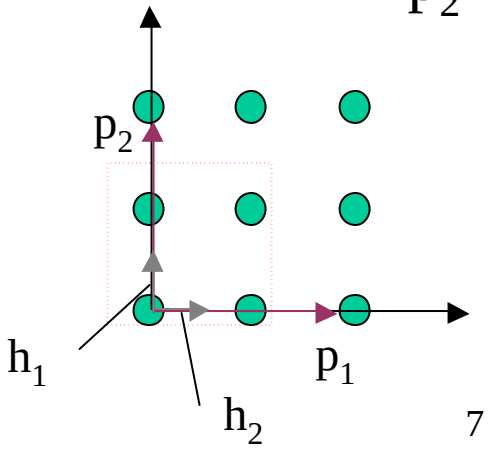
    $a(j_1, j_2) = a(j_1\text{-}1, j_2) + a(j_1\text{-}1, j_2\text{-}1)$;



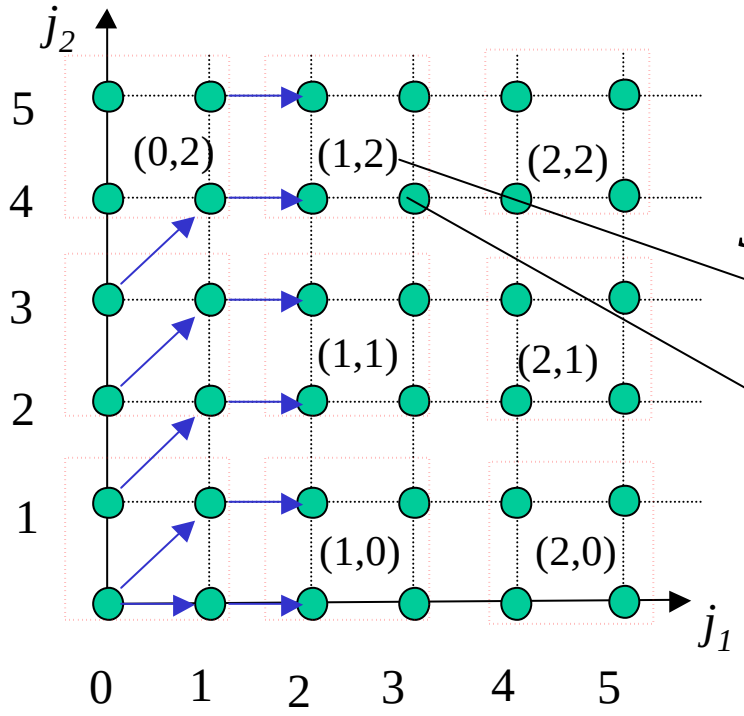
$$J^2 = \{(j_1, j_2) \mid 0 \le j_1, j_2 \le 5\}$$







Example (cont.)



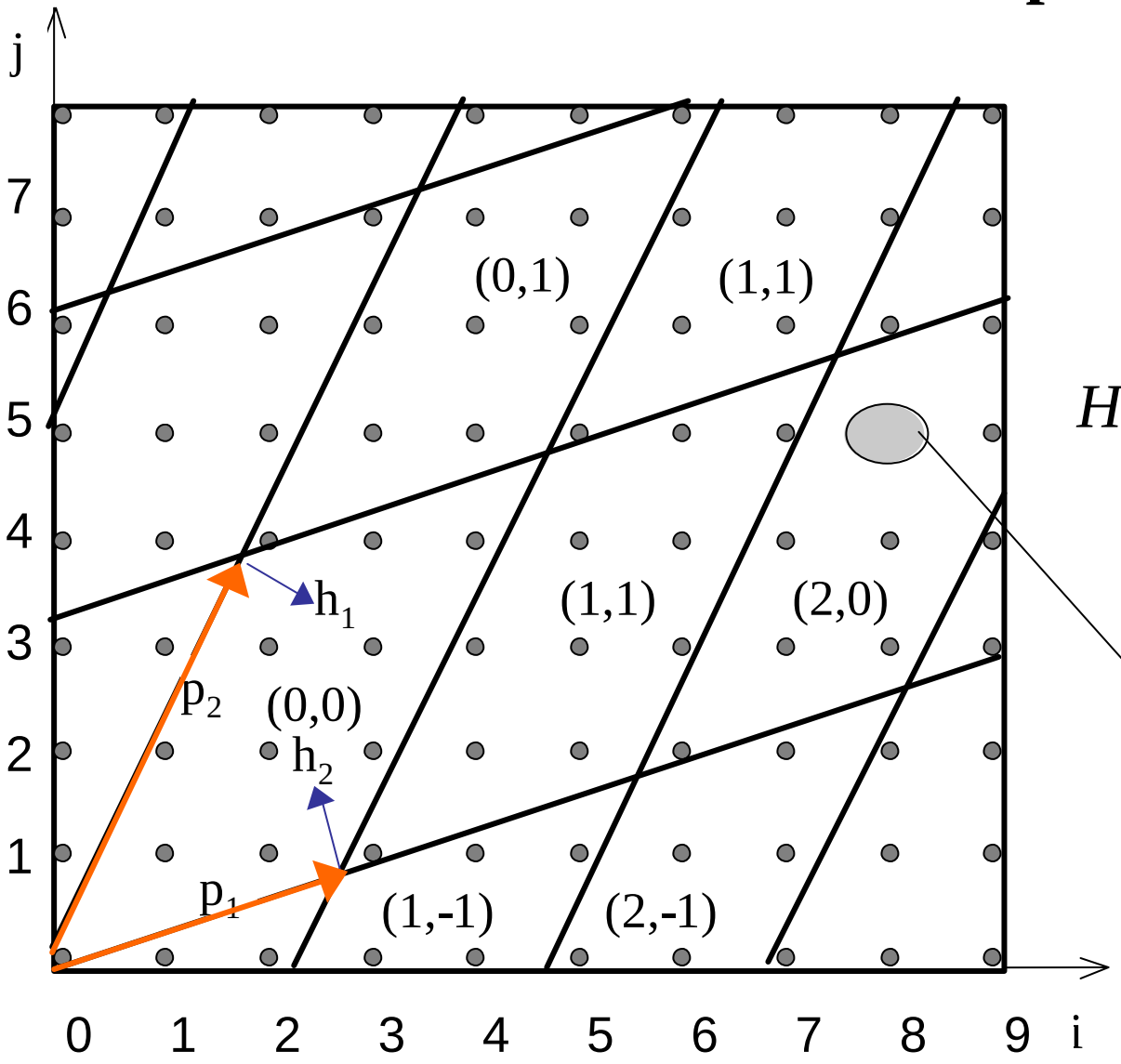
$$J^2 = \{(j_1, j_2) \mid 0 \le j_1, j_2 \le 5\}$$

$$J^S = \left\{ j^S \mid j^S = \lfloor Hj \rfloor = \begin{bmatrix} \frac{1}{2} j_1 \\ \frac{1}{2} j_2 \end{bmatrix}, j \in J^2 \right\}$$

$$r\left(\begin{bmatrix} 3 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} (1 & 2) \\ \hline (1 & 0) \end{bmatrix}$$

$$TOS(J^S, H^{-1}) = \left\{ j \in Z^n \mid j = H^{-1} j^S = \begin{bmatrix} 2 j_1^S \\ 2 j_2^S \end{bmatrix}, j^S(j_1^S, j_2^S) \in J^S \right\}$$

Another Example:



$$P = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$$

$$H = \frac{1}{10}\begin{bmatrix} 4 & -2 \\ -1 & 3 \end{bmatrix}$$

$$r\left(\begin{bmatrix} 8 \\ 5 \end{bmatrix}\right) = \begin{bmatrix} 2 \\ 0 \\ \bar{2} \\ 3 \end{bmatrix}$$

Tile Computation - Communication Cost

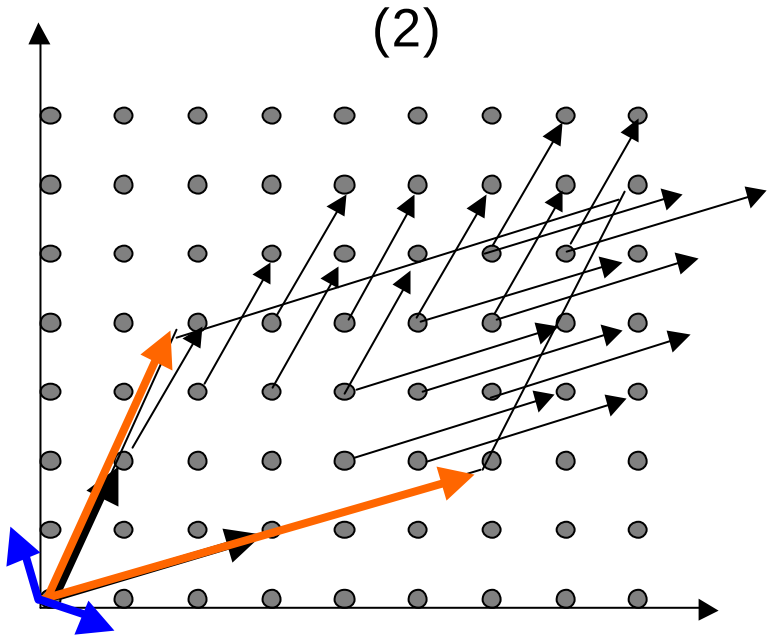
The number of iteration points contained in a supernode j^S expresses the *tile computation cost.*

The *tile communication cost* is proportional to the number of iteration points *that need to send data to neighboring tiles*

$$Minimise\ V_{comm}(H) = \frac{1}{|\det(H)|} \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{m} h_{i,k} d_{k,j}$$

$$Subject\ to\ V_{comp}(H) = \frac{1}{|\det(H)|} = n$$

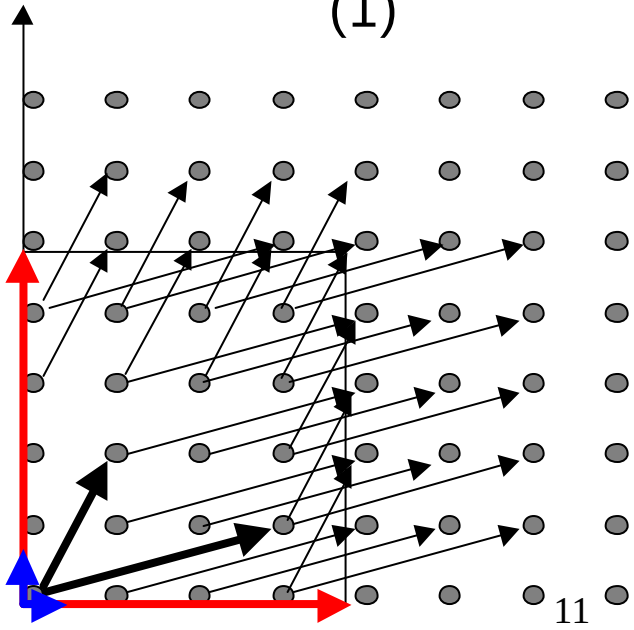
$$HD \geq 0$$



$$D = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}, P_1 = \begin{bmatrix} 4 & 0 \\ 0 & 5 \end{bmatrix}, P_2 = \begin{bmatrix} 6 & 2 \\ 2 & 4 \end{bmatrix}$$

$$V_{comp1} = V_{comp2} = 20$$

$$V_{comm1} = 27,\ V_{comm2} = 19$$



Objectives when Tiling for Parallelism

Most methods try to:

Given a computation tile volume, try to minimize the communication needs

Re-shape Tiles = reduce communication

But, how about iteration space size and boundaries?

Objective is **to minimize overall execution time**

….thus we **need efficient scheduling**

Scheduling of Tiles

If $HD \geq 0$, tiles are atomic and preserve the lexicographic execution ordering

*How can we schedule tiles to exploit **parallelism**?*

Use similar methods as scheduling loop iterations!

Solution: LINEAR TIME SCHEDULING of TILES

*What about **space** scheduling?*

Solution: CHAINS OF TILES TO SAME PROCESSOR

Linear Schedule

$$t_j = \left\lfloor \frac{\Pi j + t_0}{disp\Pi} \right\rfloor, where t_0 = -\min(\Pi i), i \in J^n$$

$$t_{j^S} = \left\lfloor \frac{\Pi j^S + t_0}{disp\Pi} \right\rfloor$$

Which is the optimal ? ?

For non-overlapping schedule: ? = [1 1 1...1]

For coarse grain tiles, all iteration dependencies are contained within a tile area.

Coarse grain ?

VERY FAST PROCESSORS

COMMUNICATION LATENCY

COMM TO COMP RATIO SHOULD BE MEANINGFUL

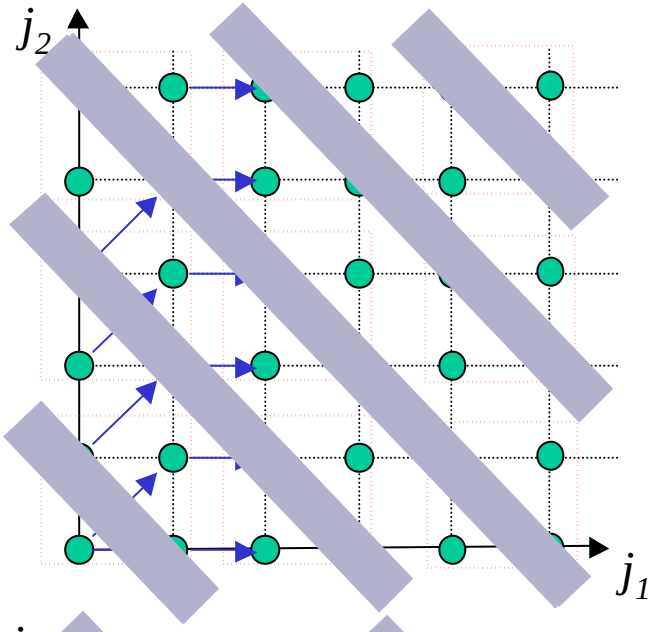
Supernode dependence set contains only unitary dependencies,

In other words, every tile communicates with its neighbors, one at each dimension

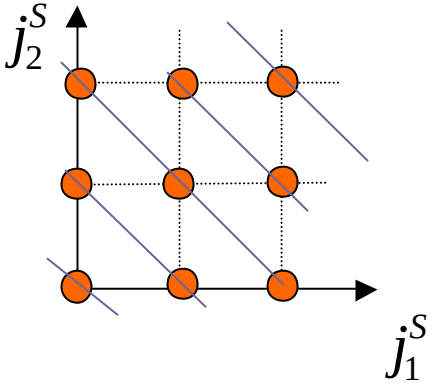
For these unitary inter-tile dependence vectors:

Optimal ? is [1 1 1…1]

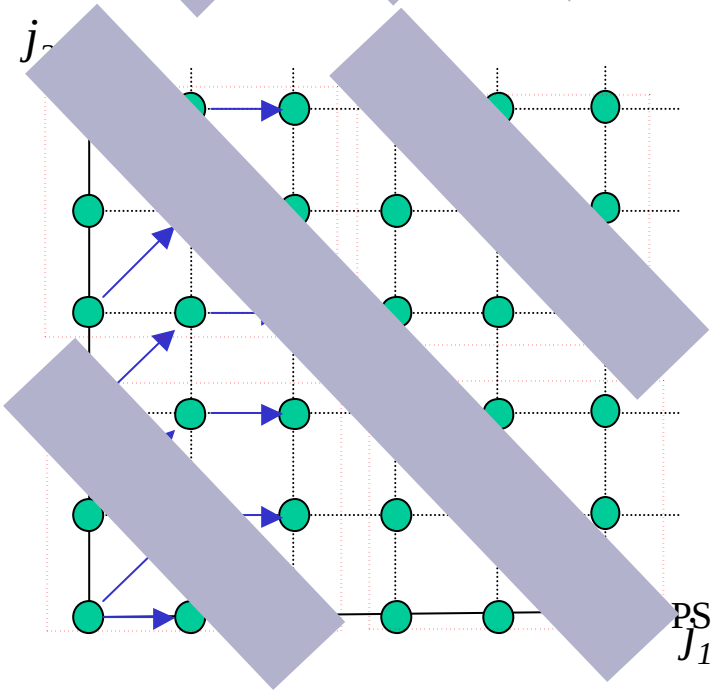
? he total number P of time hyperplanes depends on g:



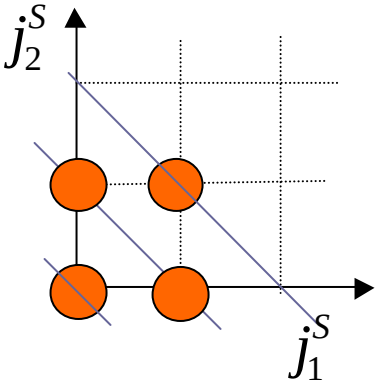
$$H = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$



Tile grain: $g = |H^{-1}| = 4$



$$H' = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$$



Tile grain: $g' = |H'^{-1}| = 9$

Each tile execution phase involves two sub-phases:

a) compute and

b) communicate results to others

How many such phases?

P(g), where P(g) the number of hyperplanes

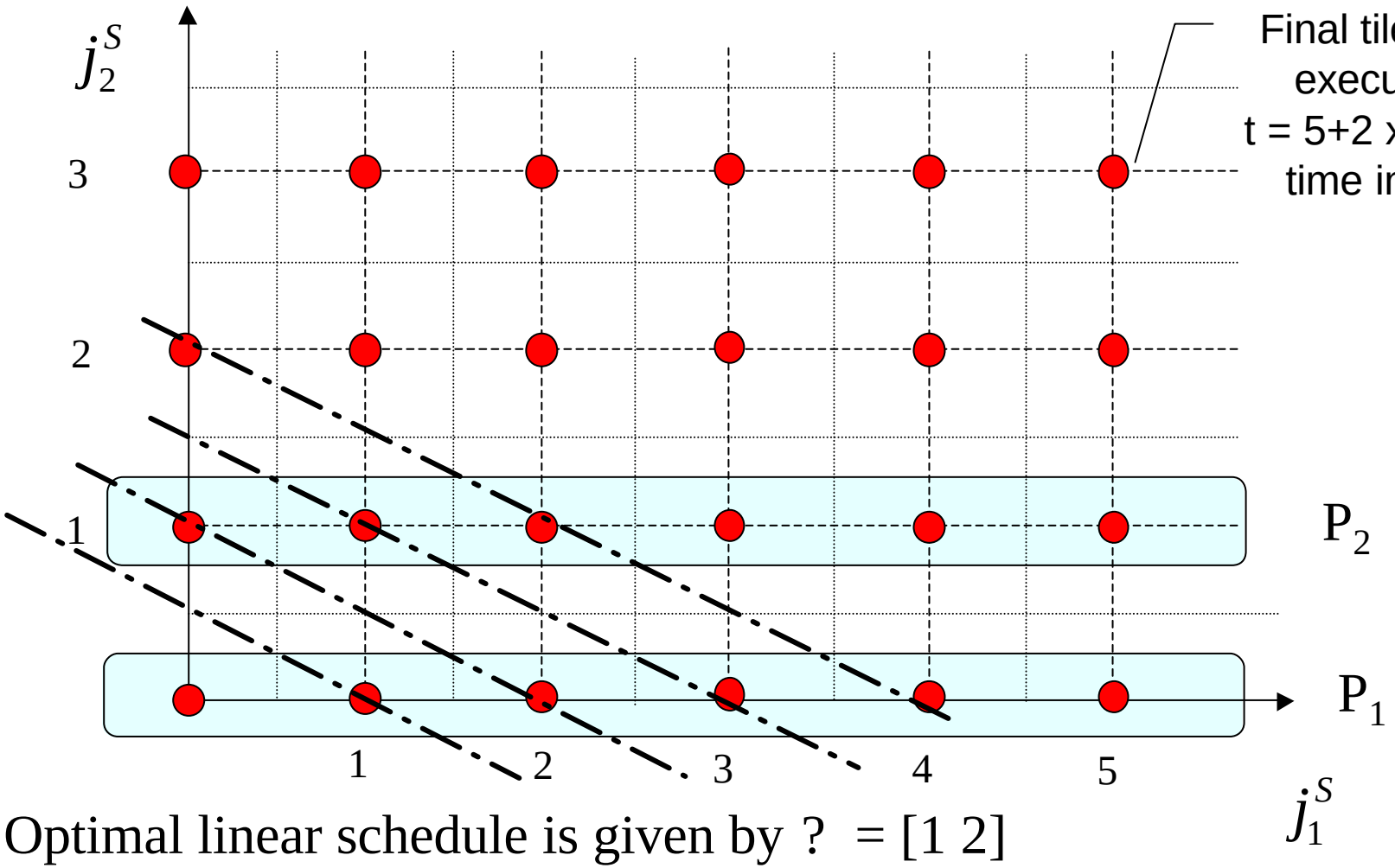
total execution time: $T = P(g)\ (T_{comp}+T_{comm})$, where:

$T_{comp}=gt_c$ the overall computation time for all iterations within a tile

T_{comm} : the communication cost for sending data to neighboring tiles

$$T_{comm}=T_{startup}+T_{transmit}$$

Mapping along the maximal dimension j_1^S :



Optimal linear schedule is given by ? = [1 2]

$$\text{For a tile } j^S(j_1^S, j_2^S),\ t_{j^S} = 2j_2^S + j_1^S + 1$$

Unit Execution Time – Unit Communication Time GRIDS

GRIDS are task graphs with unitary dependencies ONLY!

Optimal time schedule for UET-UCT GRIDS is found to be:

Assume each supernode is a task.

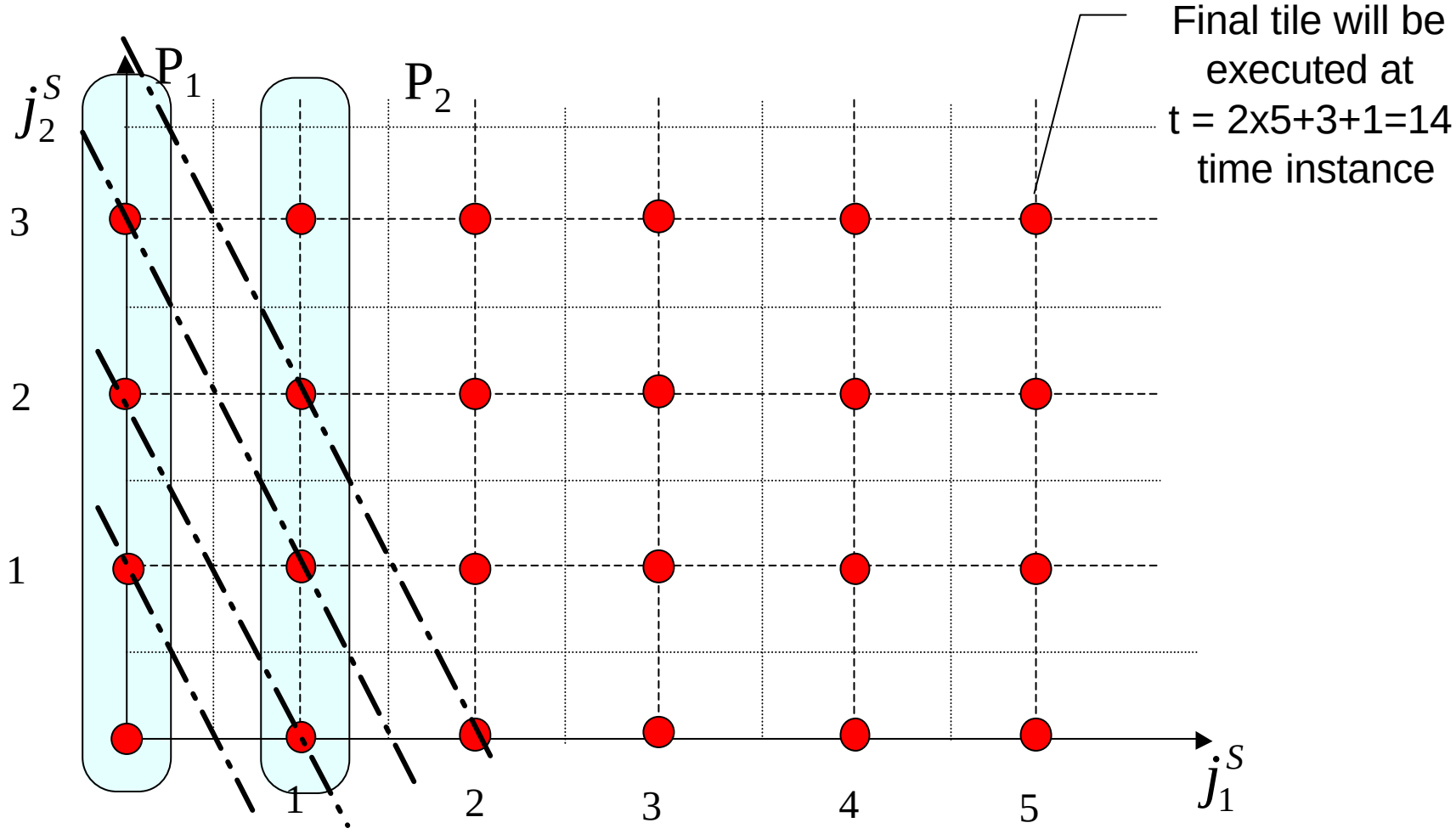
Overlapping Tile Schedule is like a UET-UCT GRID scheduling problem!

The optimal time schedule for tile $j^S\,(j_1^S, j_2^S, ..., j_n^S)$ is :

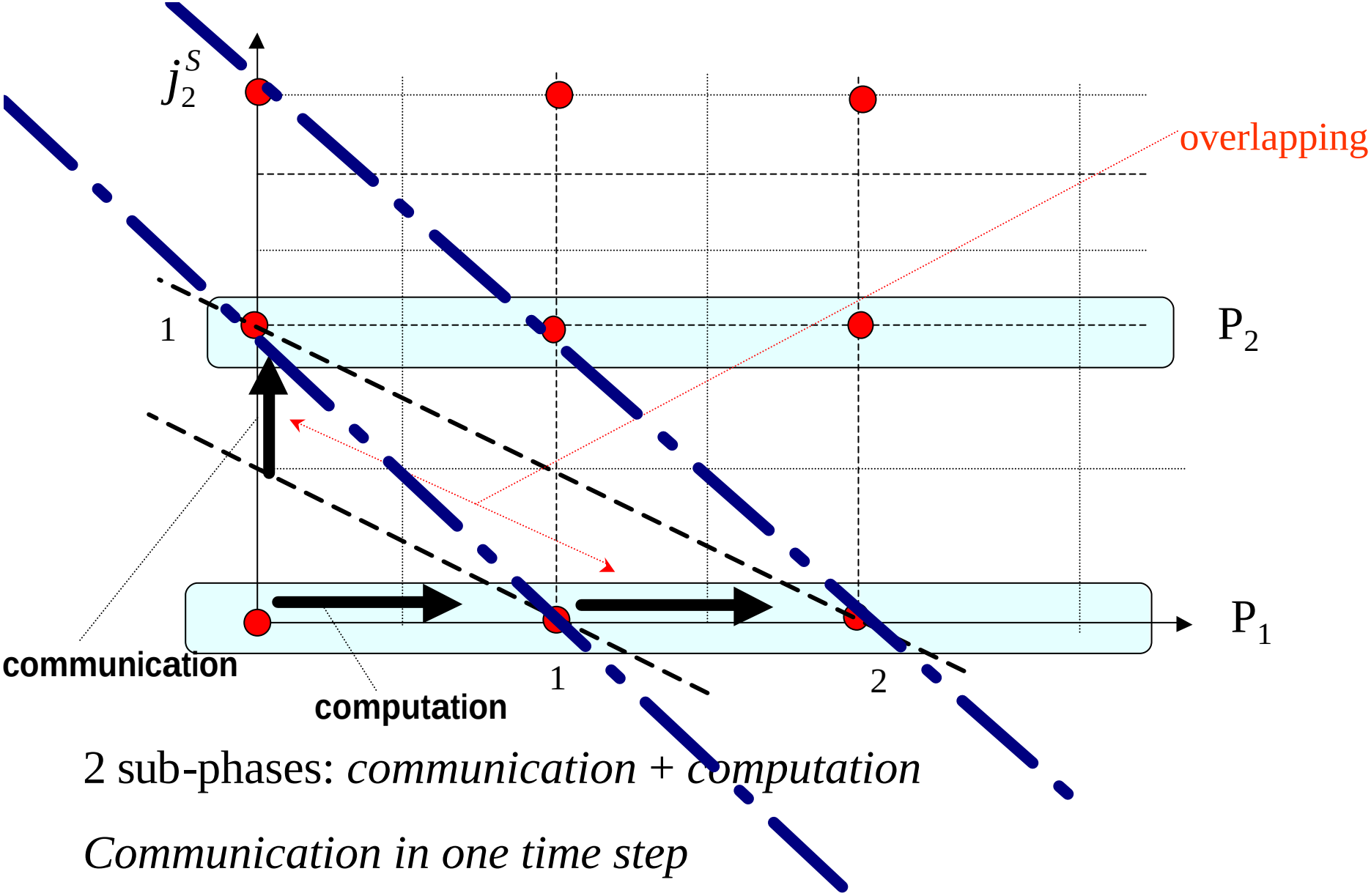
$$2\sum_{\substack{i=1 \\ i\neq k}}^{n} j_i^S + j_k^S$$, where k is the "largest" dimension

We map all tiles along k dimension to the same processor

Mapping along the non-maximal dimension j_2^S :



linear schedule now is given by ? = [2 1]. WORSE than before

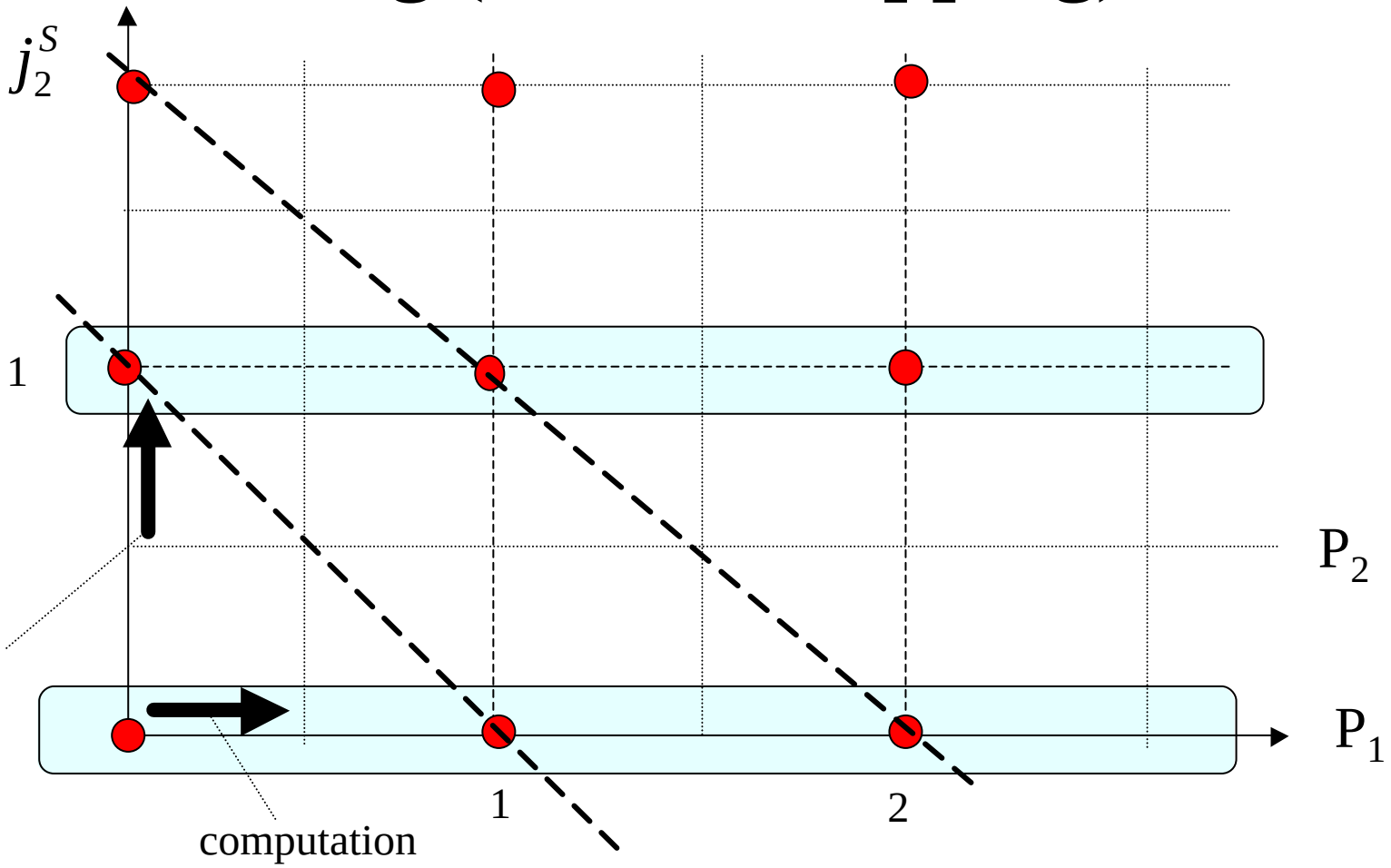


2 sub-phases: *communication + computation*

Communication in one time step

Computation in the next

Blocking (non-overlapping) case:



communication + computation in each time step

Non overlapping case

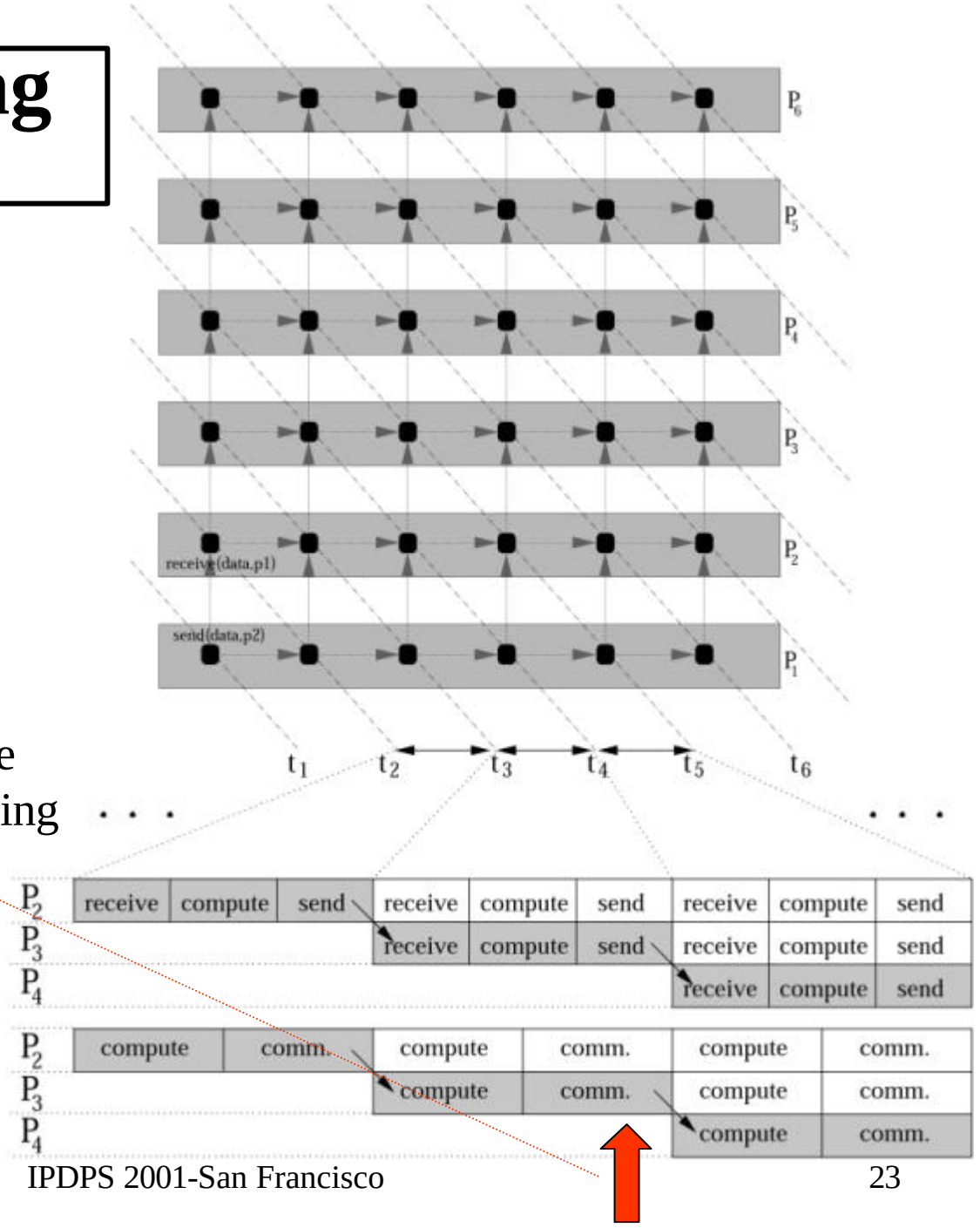
Each t*imestep* contains a *triplet* of ***receive-compute-send*** primitives

Or, equivalently:

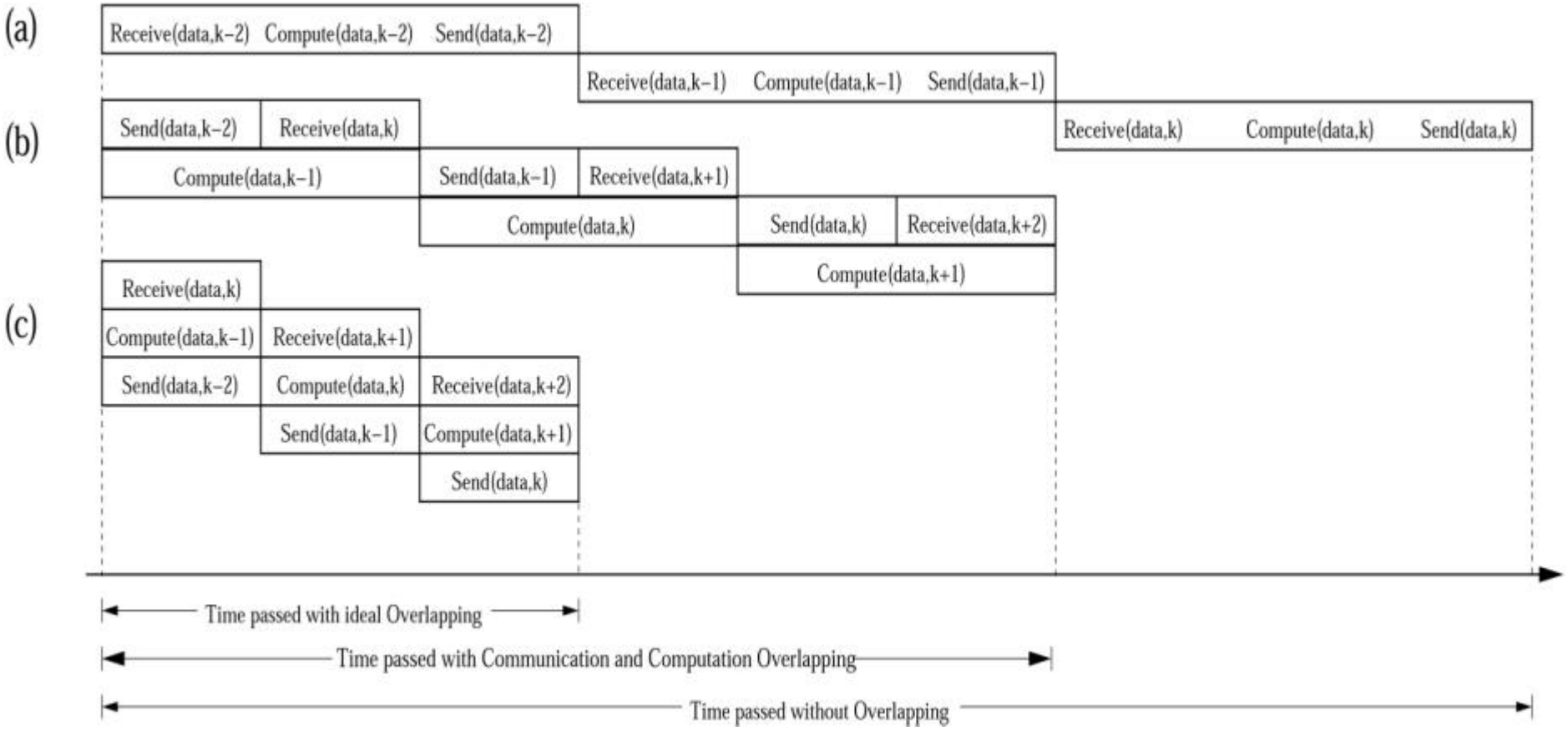
Compute-communicate

There exists time where every proc is only sending or receiving!

BAD processor utilization!



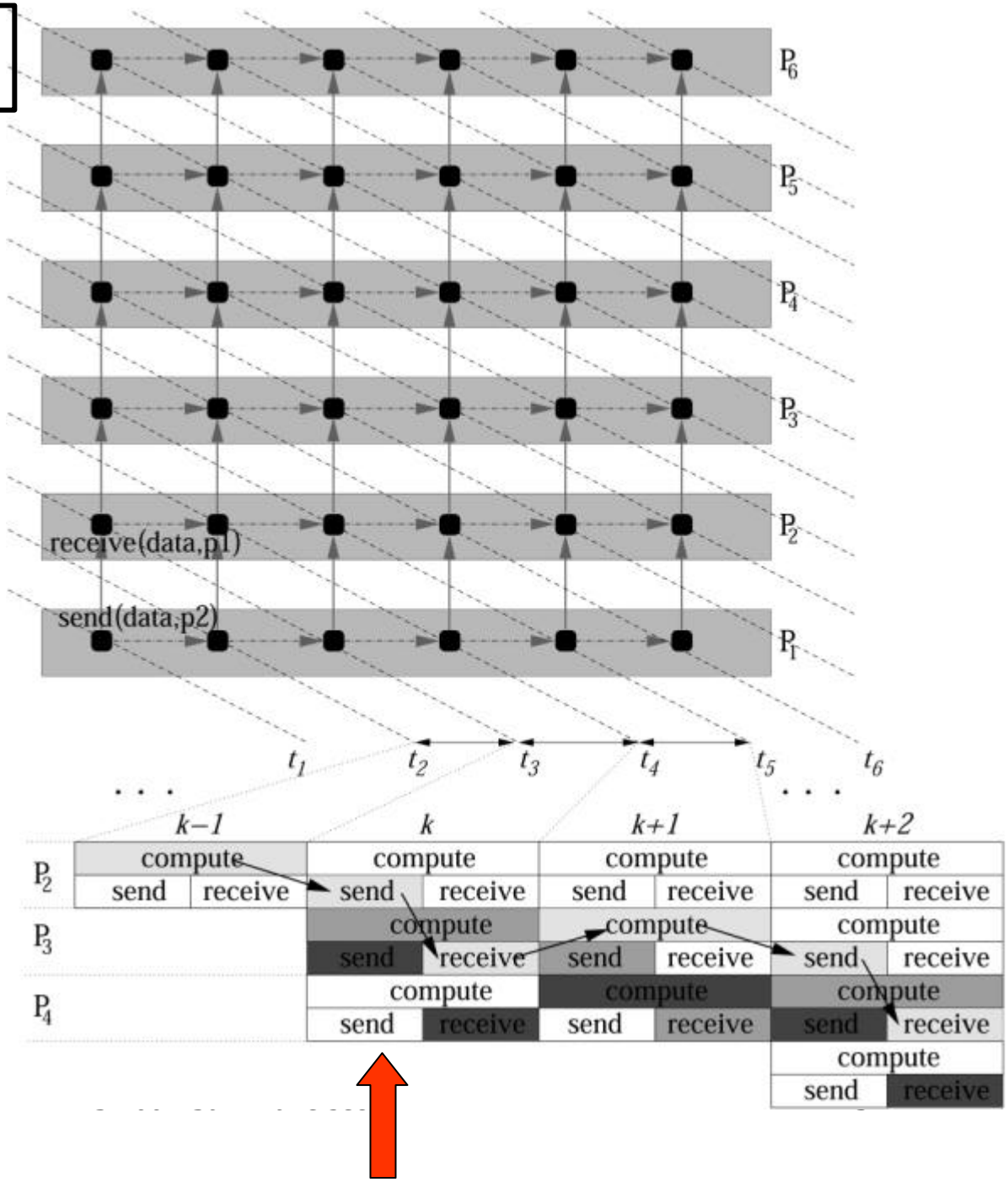
Various levels of computation to communication overlapping:



Overlapping case

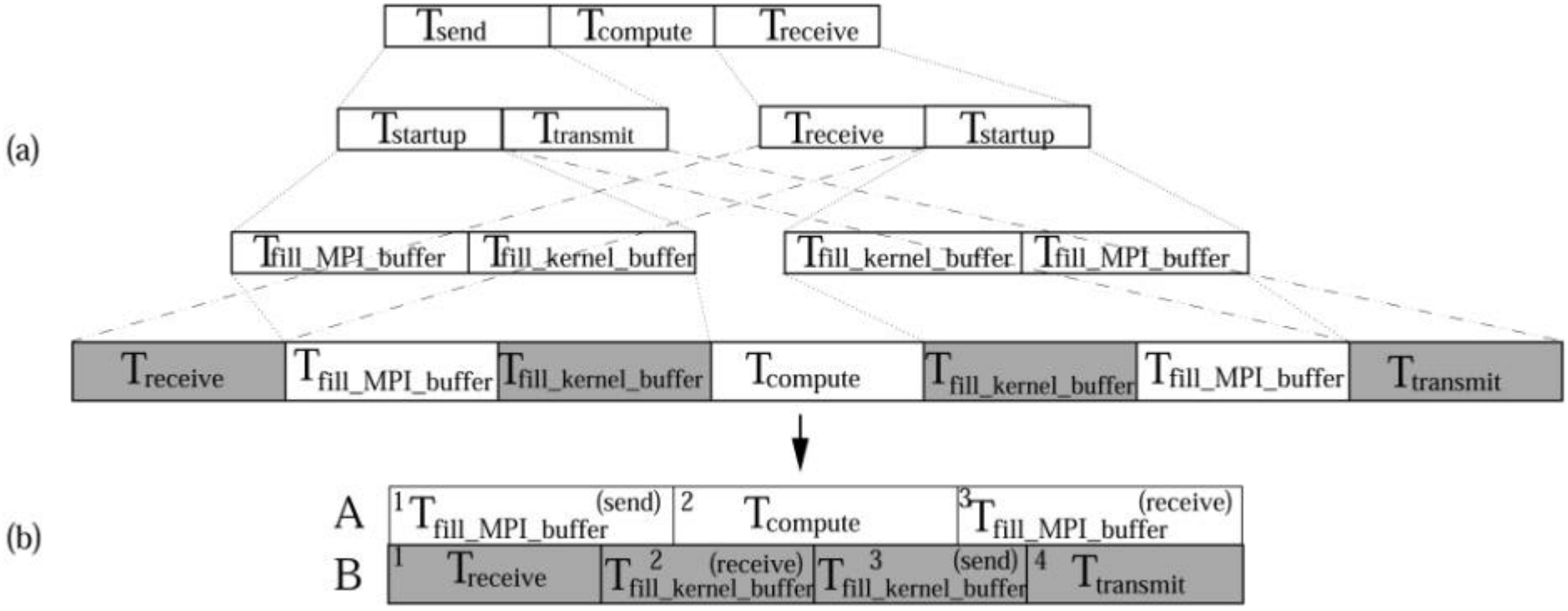
Each timestep is (ideally) **either** a **compute** or a **send+receive** primitive

Every proc computes its tile at ***k*** step and receives data to use them at ***k+1*** step, while sends data produced a ***k-1*** step



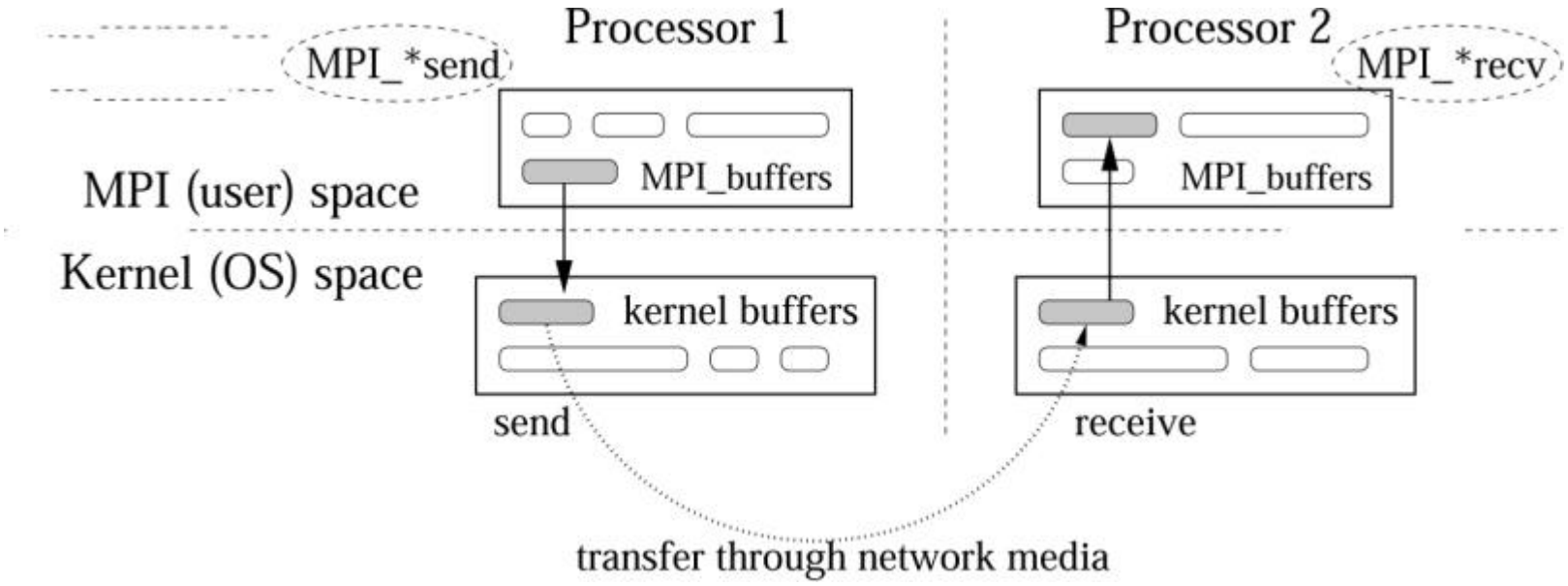
In Depth analysis of a time step

However, there exists non-avoidable startup latencies:



Thus overall time $T = P'(g) \max(A_1+A_2+A_3, B_1+B_2+B_3+B_4)$

Communication Layer Internals



Buffering + copying from user to kernel space

Sending through syscal + transmitting through media

Startup latency unavoidable (at the moment!)

But what about writing to NIC and transmitting?

(at least not the process job, but the kernel's!
Steals CPU cycles anyway!)

Experimental Results

- Linux Cluster (16 nodes + Ethernet 100Mbps + MPICH)
- Test app: *single statement triple nested loop*

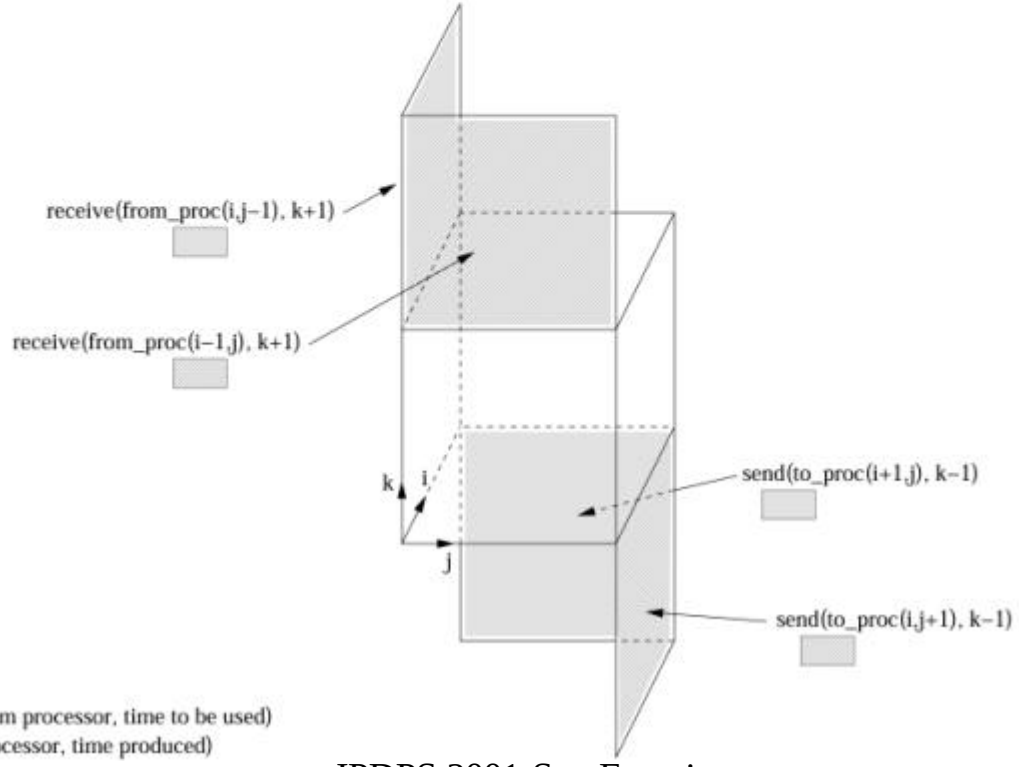
  with rectangular tiling
- k dimension is the largest one
- Each tile is a cube with *ij*, *ik* and *kj* sides
- Mapping along *k* dimension, so:

  Every processor in the *ij* plane (tile coordinates (*i*,*j*):
  1. Receives from neighbors (i-1, j) and (i, j-1)
  2. Computes
  3. Sends to neighbors (i+1, j) and (i, j+1)

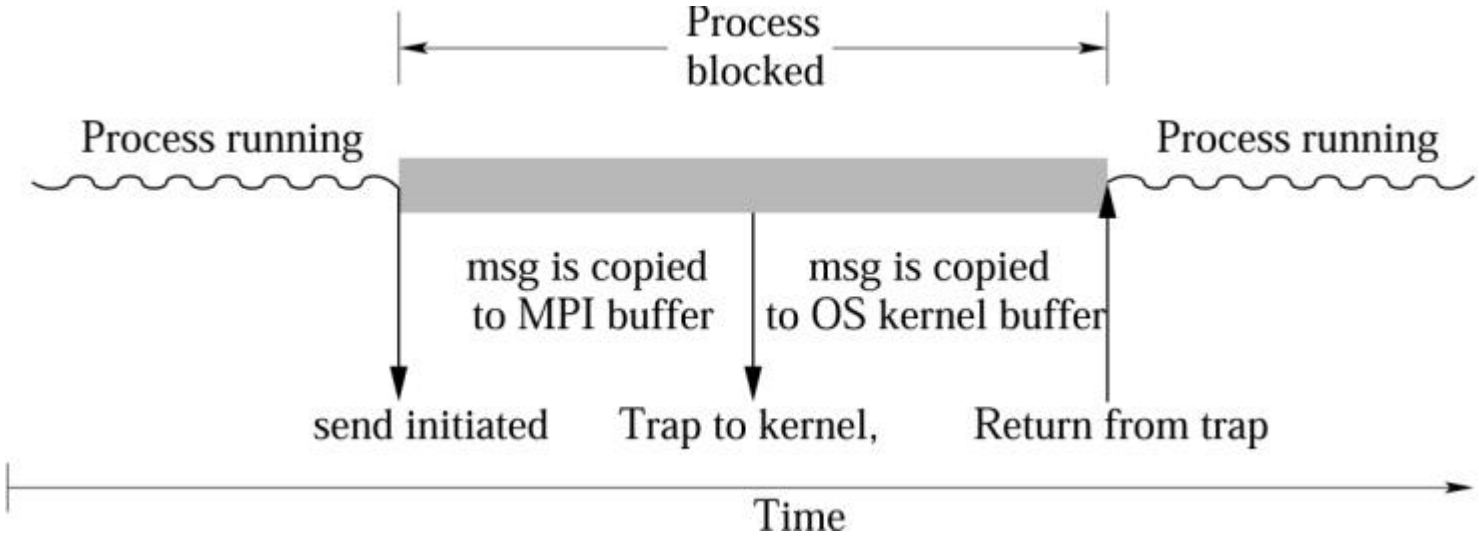
Timing and Extra buffering for the overlapping case:

TIME

k−1	k	k+1
receive(from_proc(i−1,j), k) receive(from_proc(i,j−1), k)	receive(from_proc(i−1,j), k+1) receive(from_proc(i,j−1), k+1)	receive(from_proc(i−1,j), k+2) receive(from_proc(i,j−1), k+2)
compute(proc(i,j), k−1)	compute(proc(i,j), k)	compute(proc(i,j), k+1)
send(to_proc(i+1,j), k−2) send(to_proc(i,j+1), k−2)	send(to_proc(i+1,j), k−1) send(to_proc(i,j+1), k−1)	send(to_proc(i+1,j), k) send(to_proc(i,j+1), k)



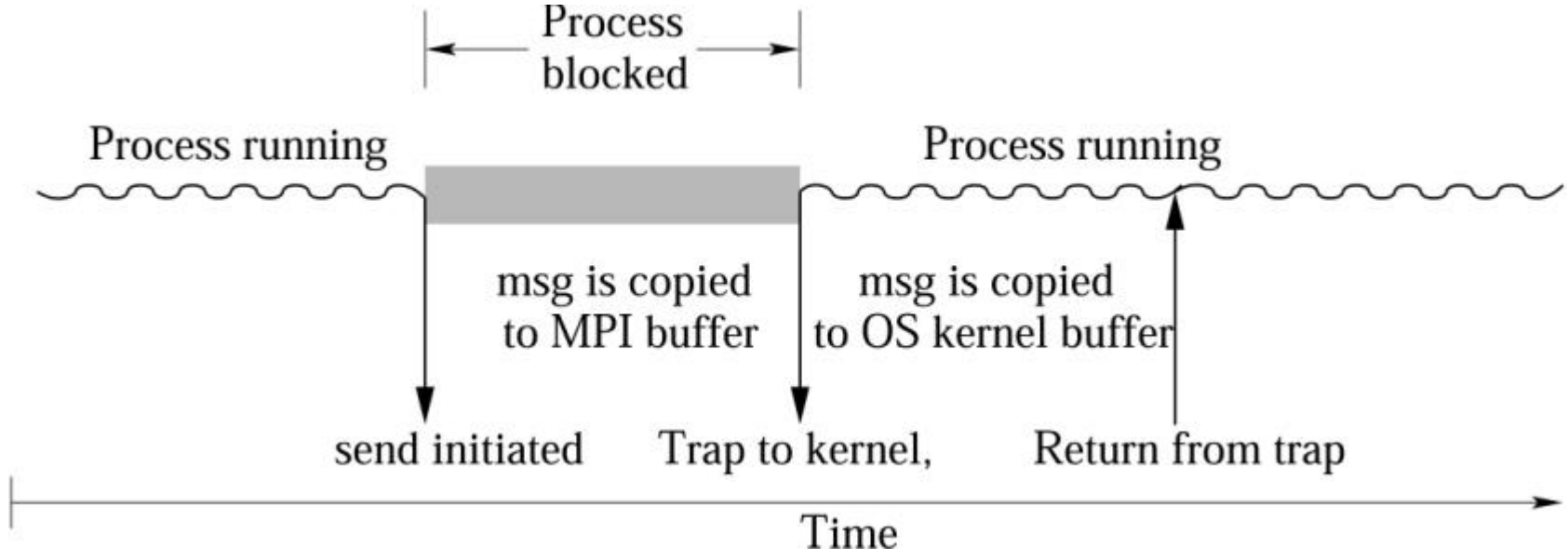
Blocking primitives



blocking case

```
For i = 0 to max_i_tile-1
For j = 0 to max_j_tile-1
    ProcB(i, j)
where ProcB(i, j) is:
  for k = 0 to max_k_tile-1
{
MPI_Recv (T(i-1, j), results (T(i-1,j), k);
MPI_Recv (T(i, j-1), results (T(i, j-1), k);
compute();
MPI_Send (T(i+1, j), results (T(i, j), k);
MPI_Send (T(i, j+1), results (T(i, j), k);
}
```

Non-blocking primitives



non-blocking case

```
For i = 0 to max_i_tile-1
For j = 0 to max_j_tile-1
    ProcNB(i, j)
where ProcNB(i, j) is:
  for k = 0 to max_k_tile-1
{
MPI_Isend (T(i+1, j), results (T(i, j), k-1).&s1);
MPI_Isend (T(i, j+1), results (T(i, j), k-1), &s2);
MPI_Irecv (T(i-1, j-1), results (T(i-1, j), k+1), &r1);
MPI_Irend (T(i, j-1), results (T(i, j-1), k+1), &r2);
compute();
MPI_wait(s1); MPI_wait(s2);
MPI_wait(r1); MPI_wait(r2);
}
```

AxBxC (i, j, k) iteration spaces

Use 16 processors: 4 processor in each dim i, j

16 x16 x 16384, 16 x 16 x 32768, 32 x 32 x 4096

Tiles of size 4x4xV, 8x8xV, for variable V, thus variable g

Methodology:

Find $V_{experimental}$, $g_{experimental}$ for which T_{min}

Calculate t_c (computation for one iteration)

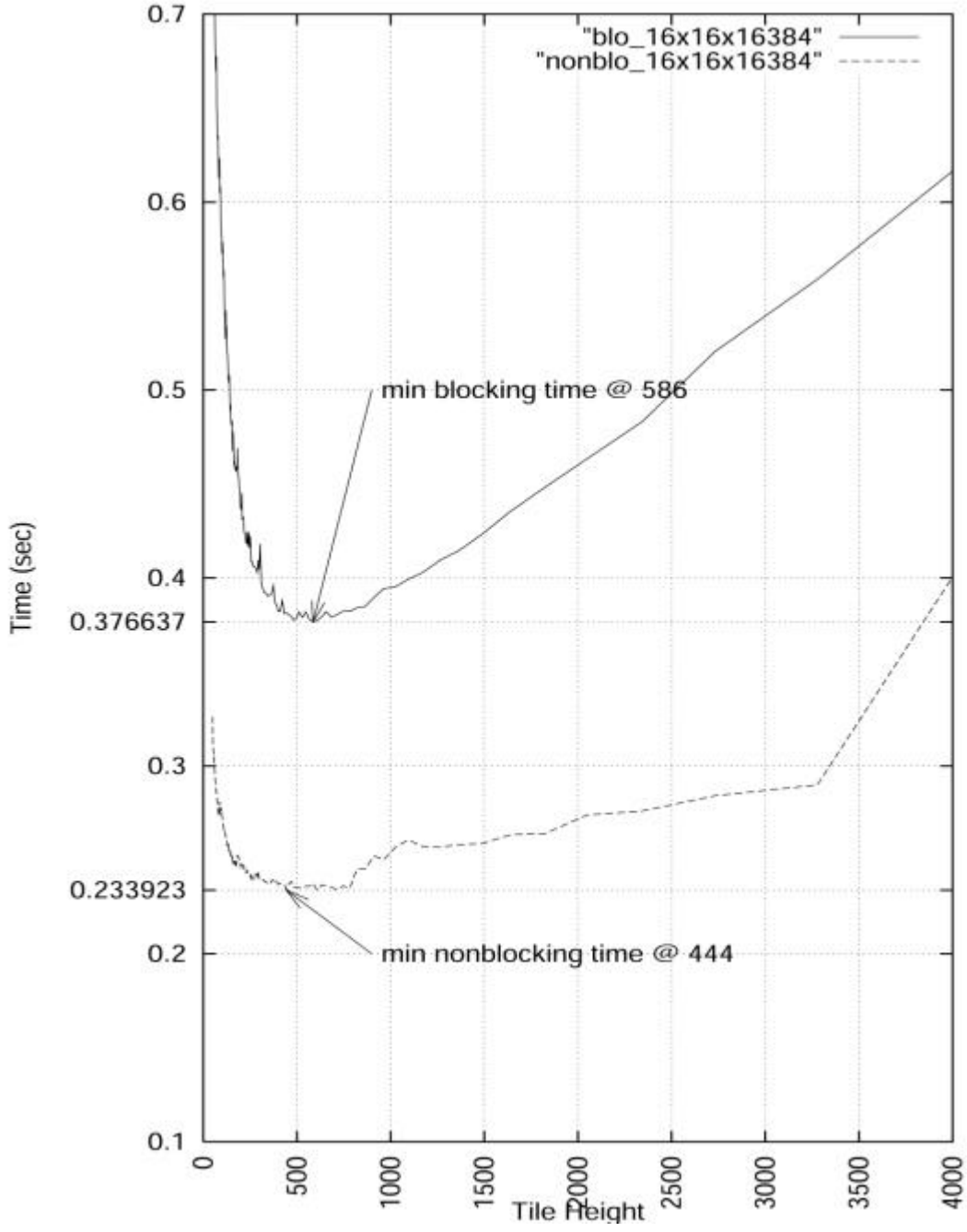
Calculate $T_{fill\_MPI\_buffer}$ experimentally for $V_{experimental}$

Which is $P(g_{experimental})$ (# of hyperplanes)?

Find by formula $T_{theoret}$ using $P(g_{experimental})$

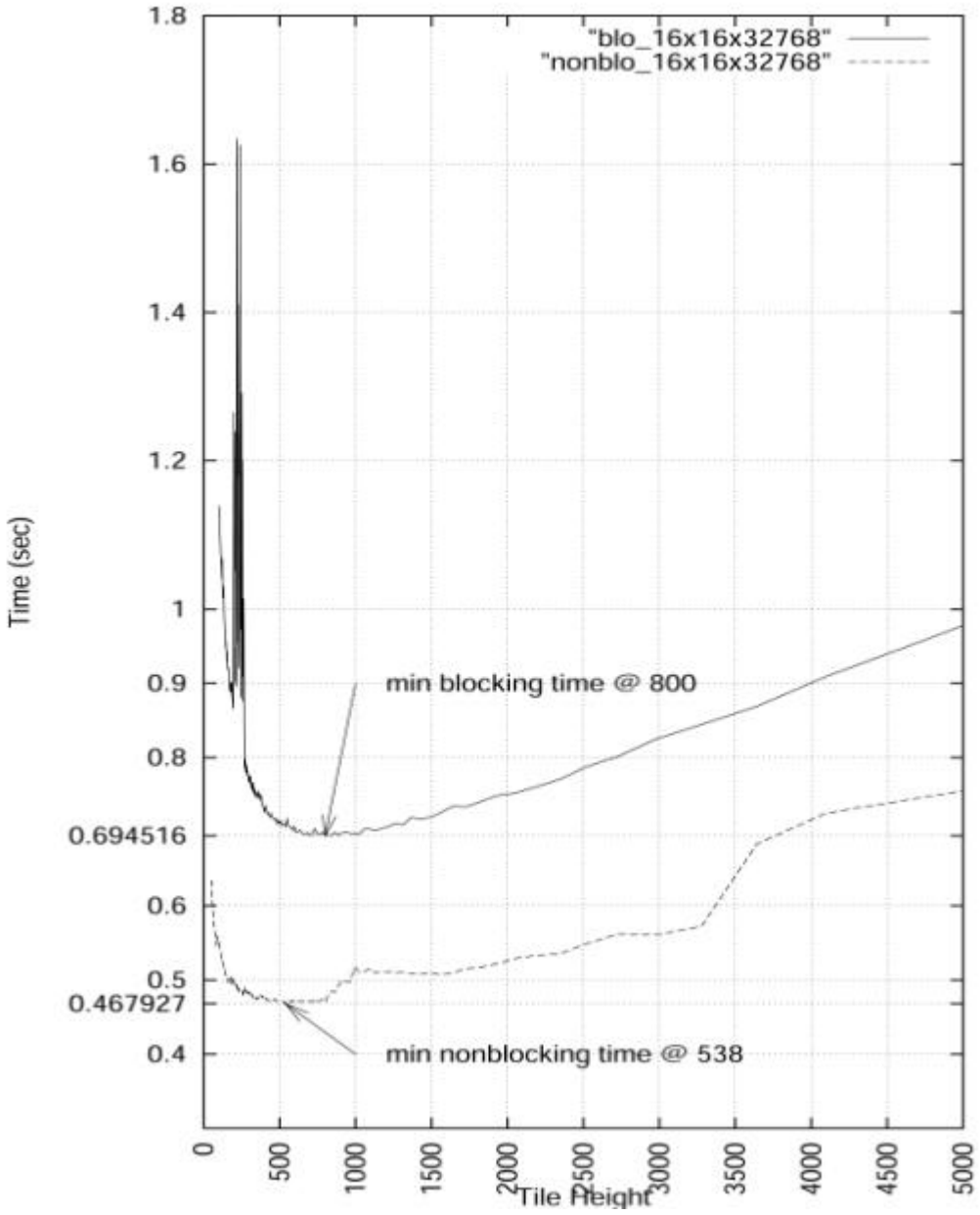
Compare T_{min} and $T_{theoret}$

16_16_16384



II

16_16_32768


"blo_16x16x32768"
"nonblo_16x16x32768"
Time (sec)
1.8
1.6
1.4
1.2
1
0.9
0.8
0.694516
0.6
0.5
0.467927
0.4
min blocking time @ 800
min nonblocking time @ 538
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Tile Height

32_32_4096

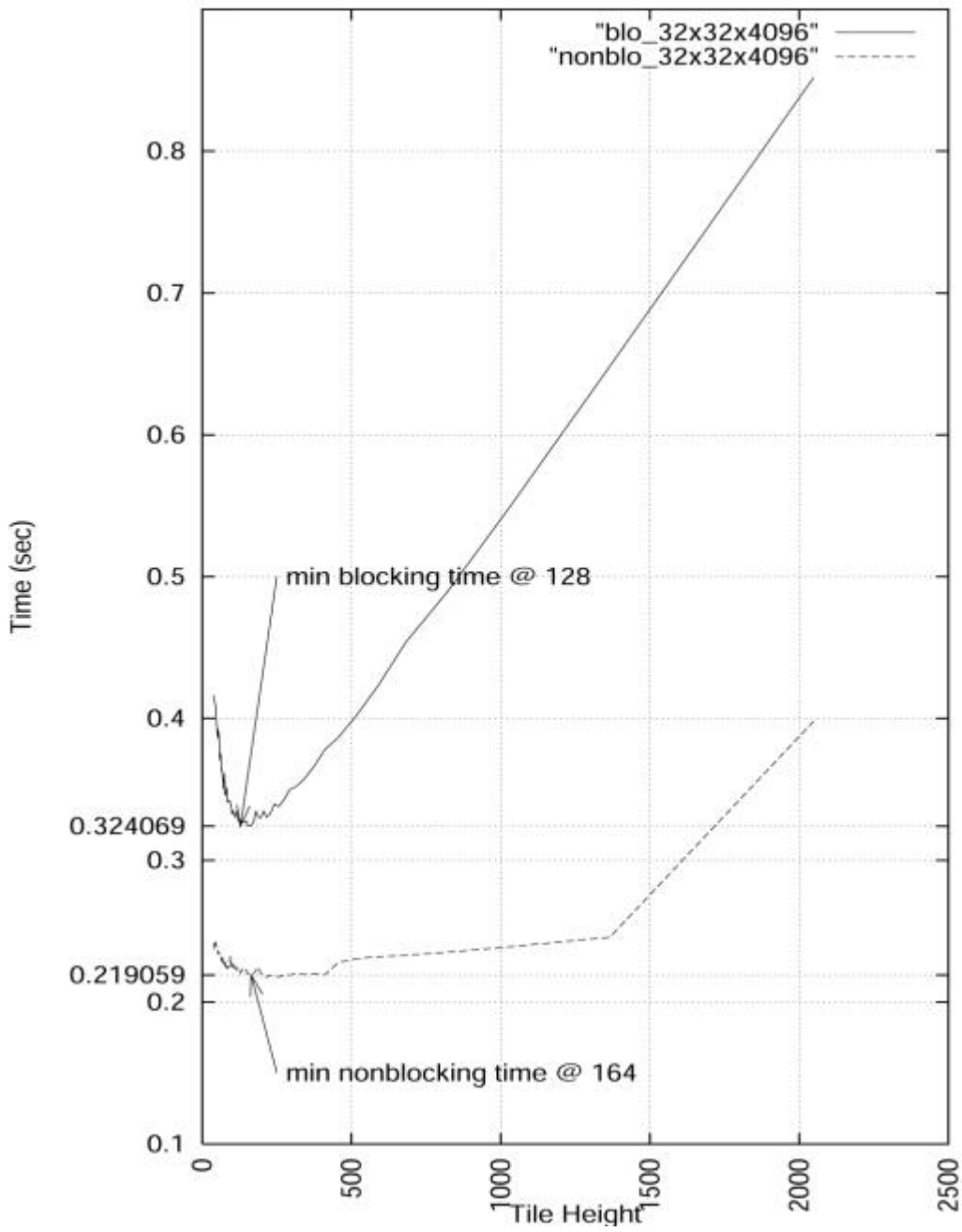

"blo_32x32x4096"
"nonblo_32x32x4096"
min blocking time @ 128
min nonblocking time @ 164
Time (sec)
Tile Height
0.8
0.7
0.6
0.5
0.4
0.324069
0.3
0.219059
0.2
0.1
0
500
1000
1500
2000
2500

Table of Results

	i	ii	iii
index set size ($i \times j \times k$)	$16 \times 16 \times 16384$	$16 \times 16 \times 32768$	$32 \times 32 \times 4096$
$V_{optimal}$	444	538	164
$g_{optimal}$	7104	8608	10996
$t_{optimal}$ overlapping experimental	0.233923 sec	0.467929 sec	0.219059 sec
$t_{fill\ MPI\ buf}$	0.627 msec	0.745 msec	0.37 msec
P(g)	53	76	41
$t_{optimal}$ overlapping theoretical	0.24 sec	0.507 sec	0.25 sec
difference experimental vs. theoretical	2.5%	7%	12%
$t_{optimal}$ non-overlapping experimental	0.376637 sec	0.694516 sec	0.324069 sec
improvement overlapping vs. non-overlapping	38%	33%	32%

Can we find analytical expressions for $A_i(g)$, $B_i(g)$?

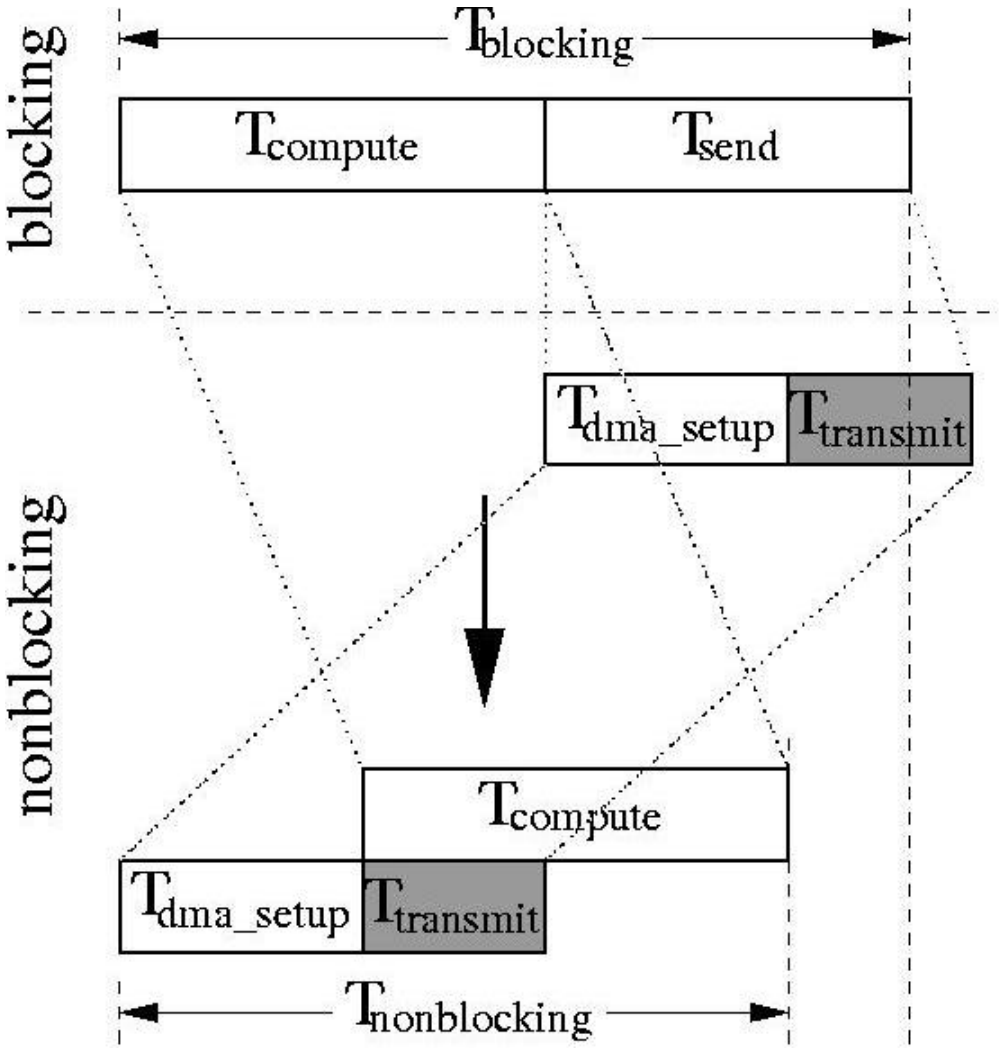
Too difficult

High level communication layers seem to abstract

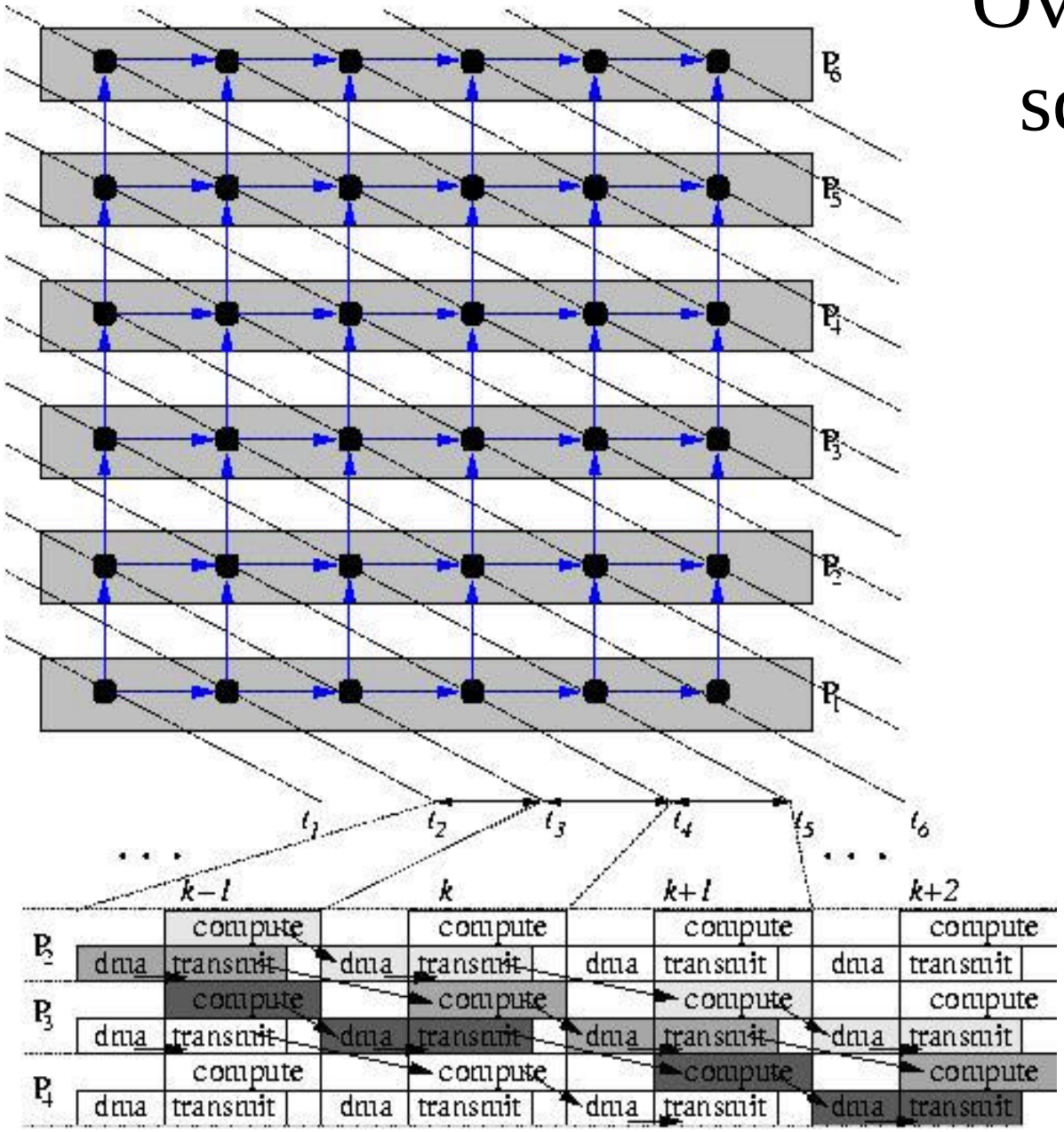
Need lower latency layers?

zero-copy protocols +DMA

Timestep Analysis using kernel level DMA



Overlapping time schedule using DMA



Kernel Level initiation of DMA

Using DMA avoids the CPU OS cycle stealing when copying from kernel space to NIC buffers

However:

When DMA is started from kernel

- OS kernel checks the size of the user memory area segment
- OS kernel translates VM to contiguous phys (DMA needs phys mem addresses)
- OS kernel writes args and size to DMA engine registers

THUS: Data are copied from user space to contiguous kernel space mem by CPU

DMA startup latency (due to OS ops) is increasing in comparison with transmission time

Solution: USER LEVEL NETWORKING LAYER

Ongoing Work

- We use SCI (Scalable Interconnection Network) with DMA capabilities (Dolphin D330 cards)
- Two threads of control per process
- CPU does very little job, thus small startup latencies (even with DMA engine startups)
- Coarser tile grains than before!

User Level Networking

AM, FM, U-NET and BIP then VIA = standard

Messages are sent directly from user space without OS intervention

User level communication endpoints

How about starting DMA from user level?

Evolution to DMA

•It would be nice if we could write from user level directly to contiguous physical memory!

mmap “RAM device”

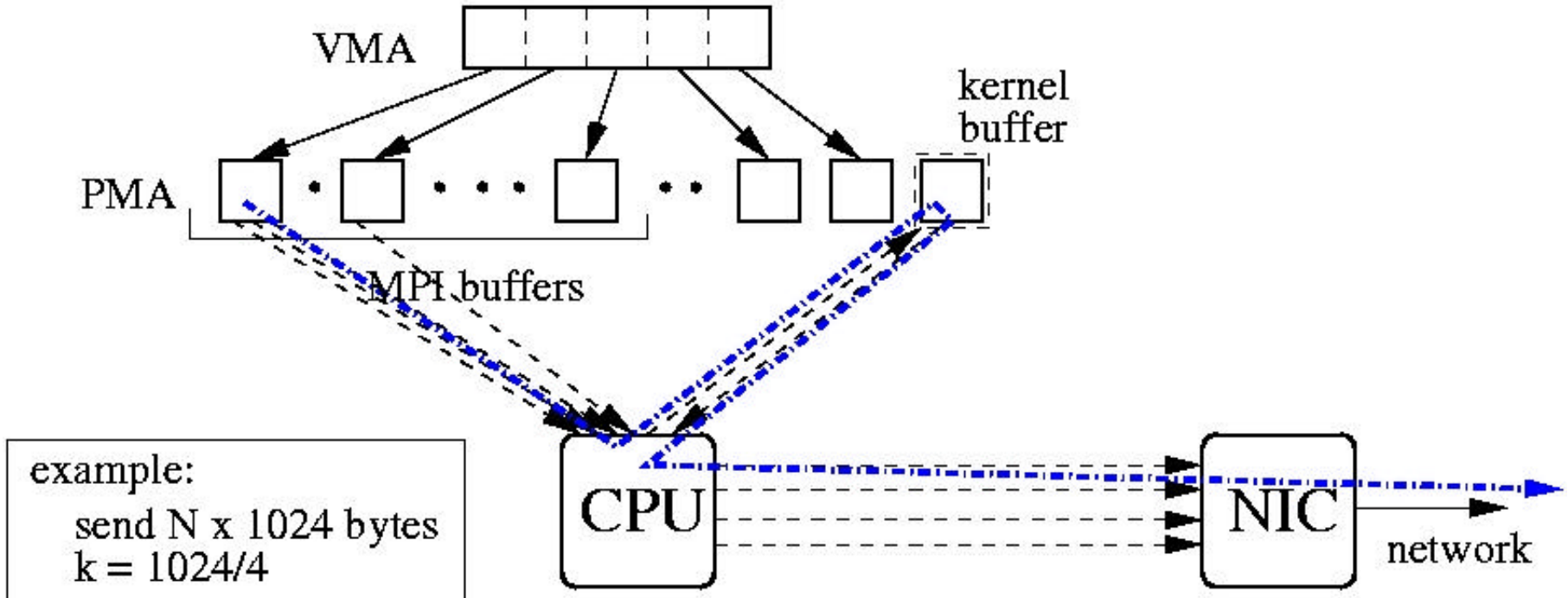
We save CPU from the copy to contiguous memory areas.

•It would be nice if we could initiate DMA from user level!

Support from OS and device

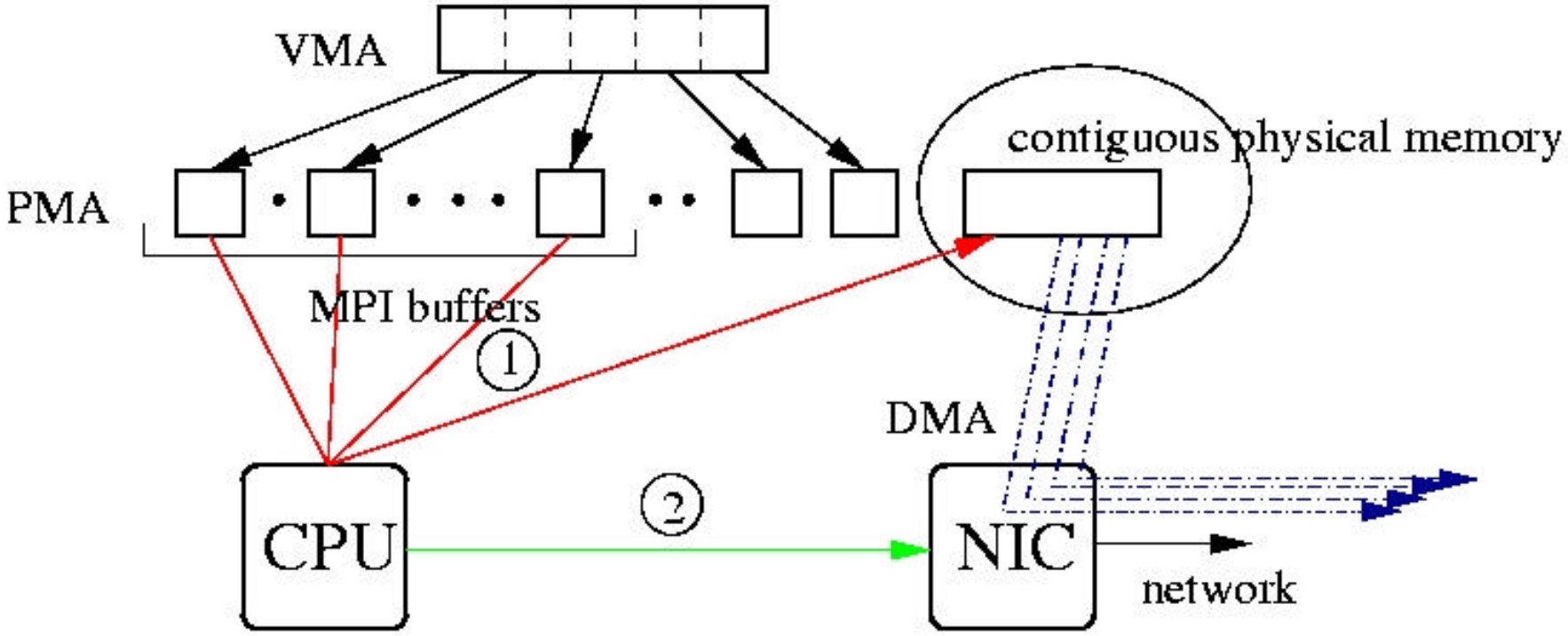
We save CPU from memory to device copy.

MPI with Ethernet simple send



CPU delay = N x [from user to kernel(i.e 1024 bytes) + k x from kernel to NIC(i.e 4 bytes)]

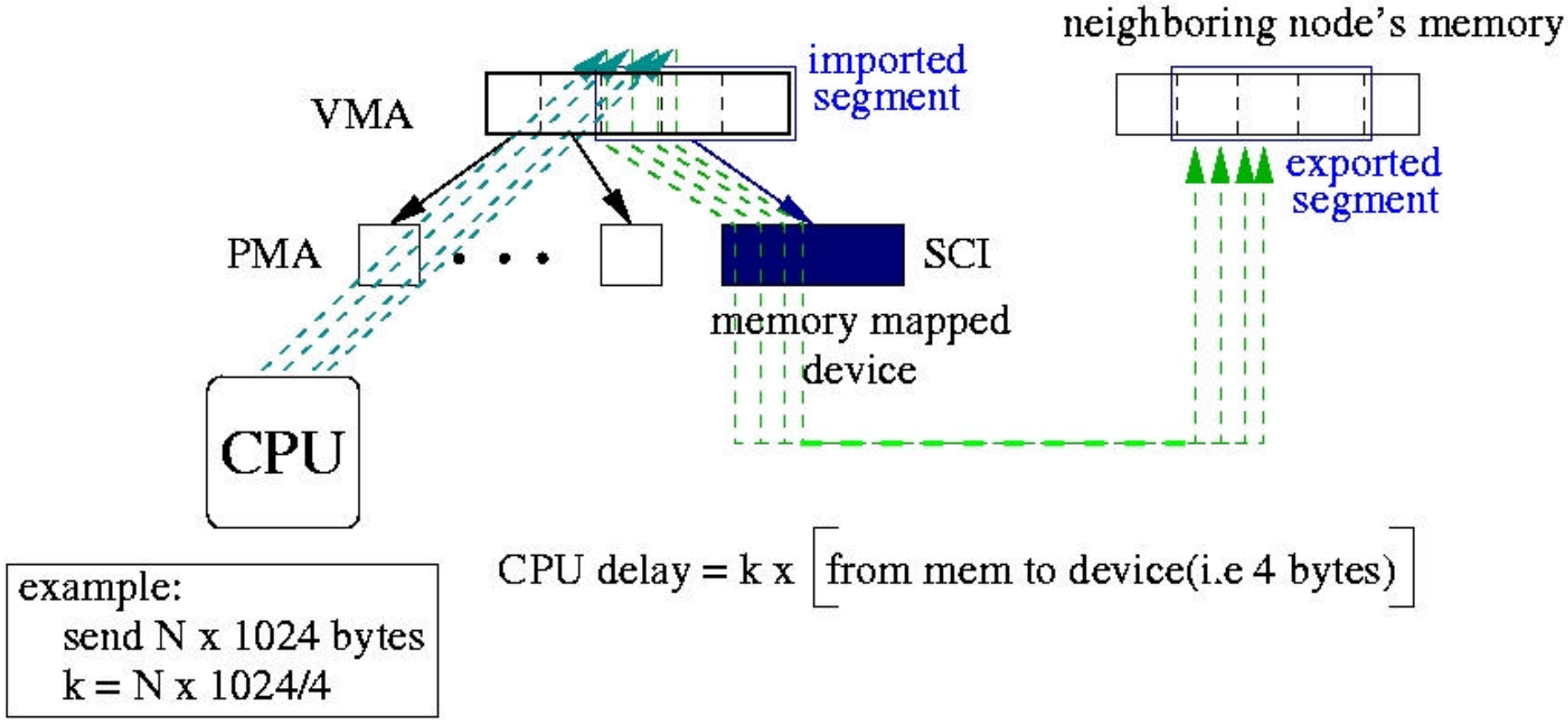
MPI with Ethernet
DMA send



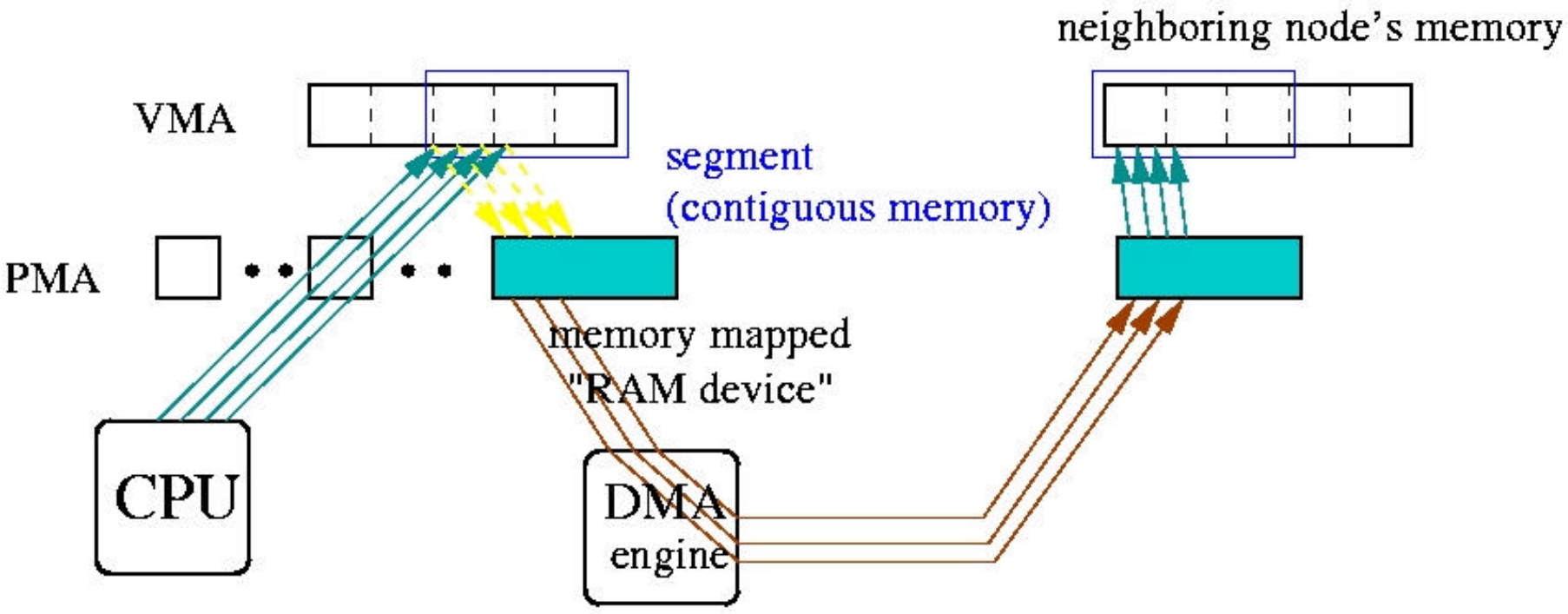
example:
send N x 1024 bytes

CPU delay = [from user to kernel (N x 1024 bytes) + startup DMA engine]

SCI with Shared Memory Send



Our approach
SCI with DMA send



example:
send N x 1024 bytes

CPU delay = DMA setup