Lecture 7:
Programming for Performance (part II)

CMU 15-418: Parallel Computer Architecture and Programming (Spring 2012)

Assignment 2

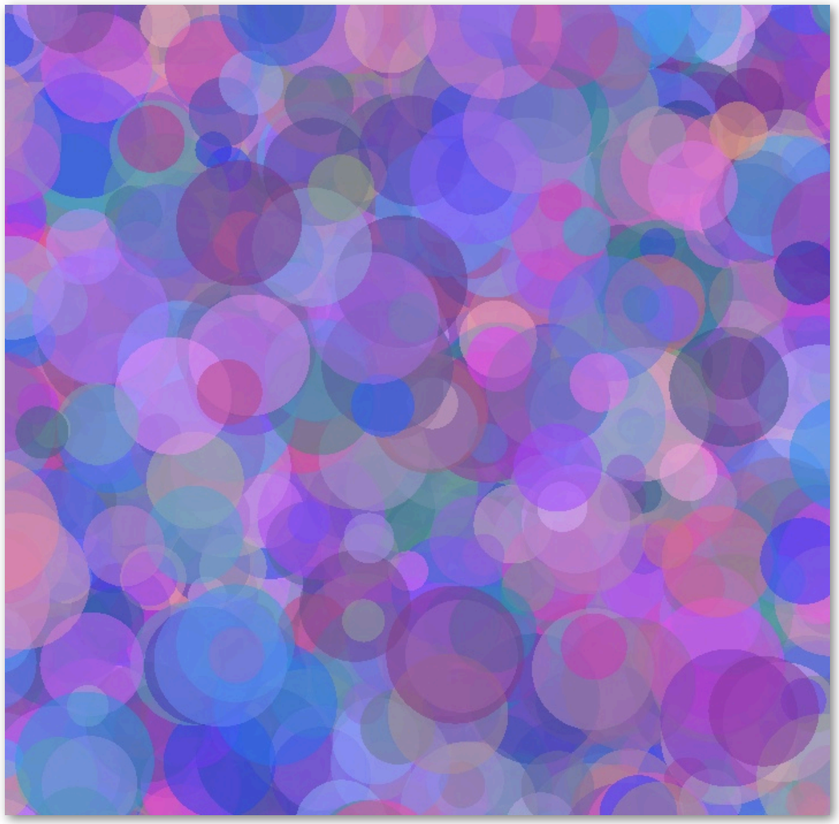
- **Assignment 2 out tonight**
  - **We are slowly bringing up machines with the new GPUs**
  - **(It requires cutting metal)**

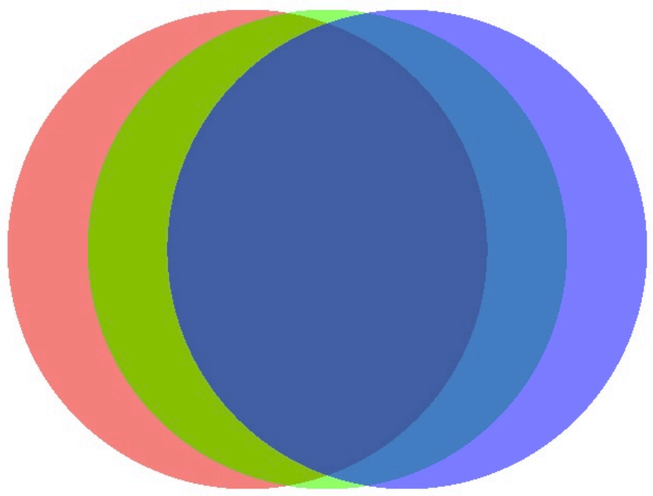
- **Due Tuesday, Feb 21 (2 weeks)**

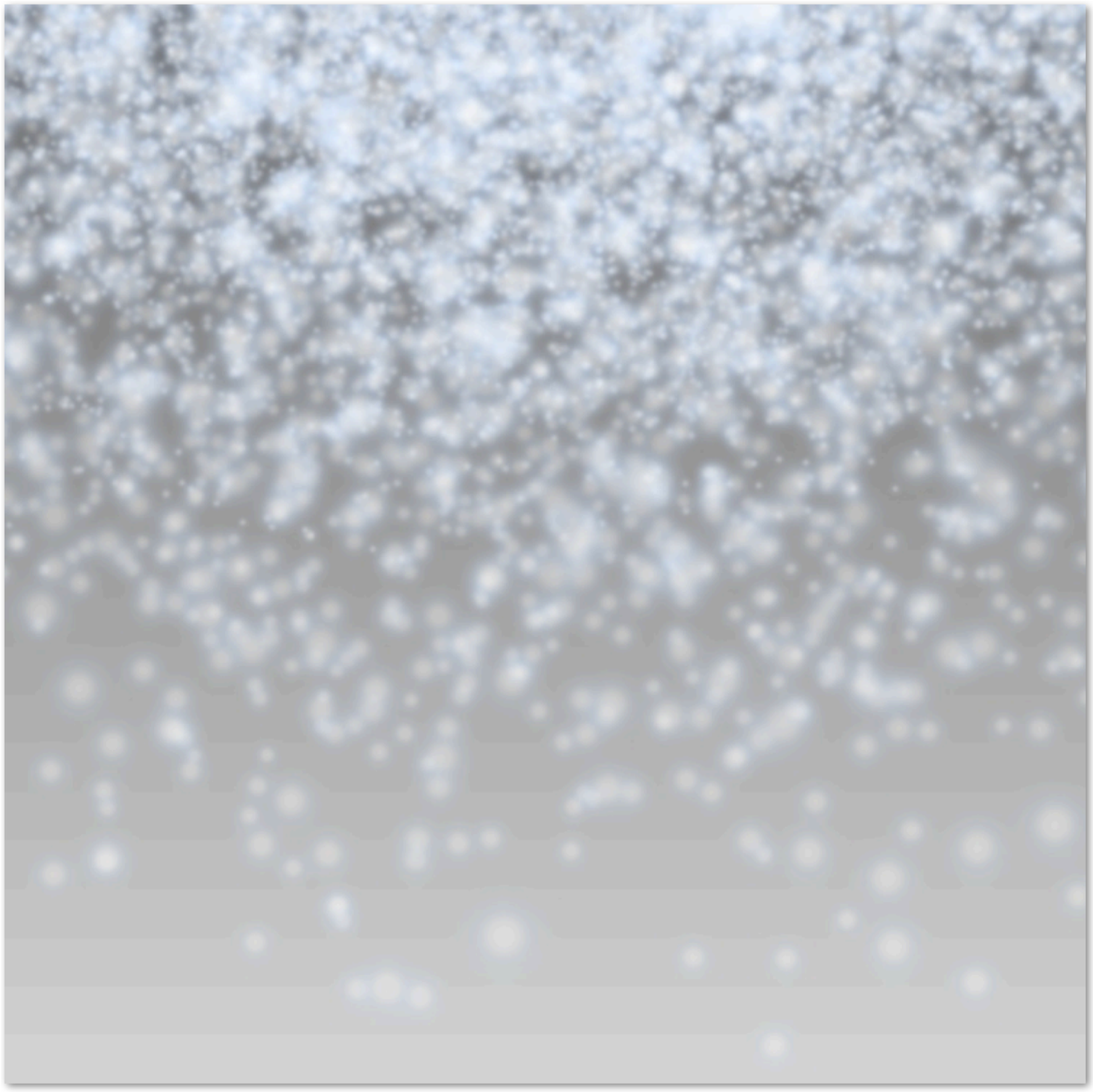
- **May work in pairs**

Assignment 2

Rendering circles





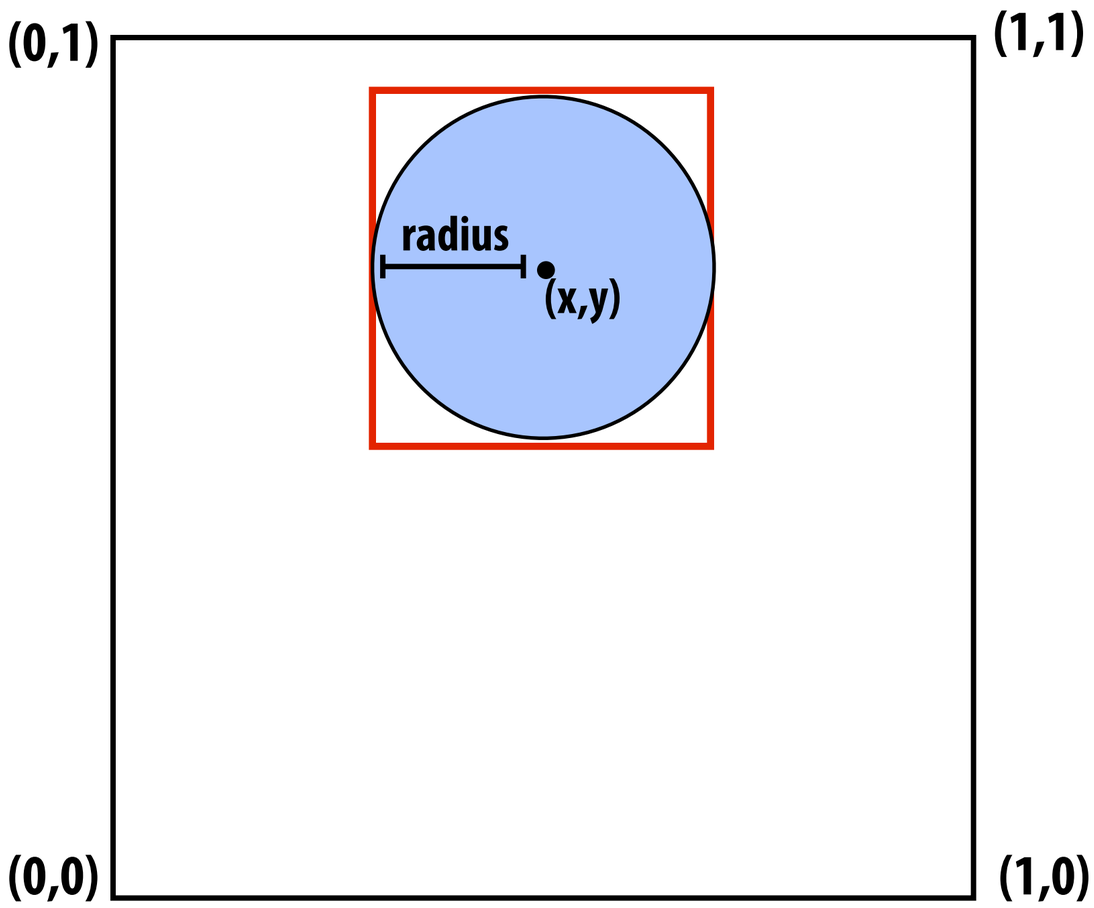


Renderer structure

```
class CircleRenderer {

public:

    virtual void clearImage() = 0;

    virtual void advanceAnimation() = 0;

    virtual void render() = 0;
};
```

Renderer structure

```
if (animating)
   for each circle
      update position and velocity

for each circle
   compute screen bounding box
   for all pixels in bounding box
      compute pixel center point
      if center point is within the circle
         compute color of circle at point
         blend contribution of circle into image for this pixel
```



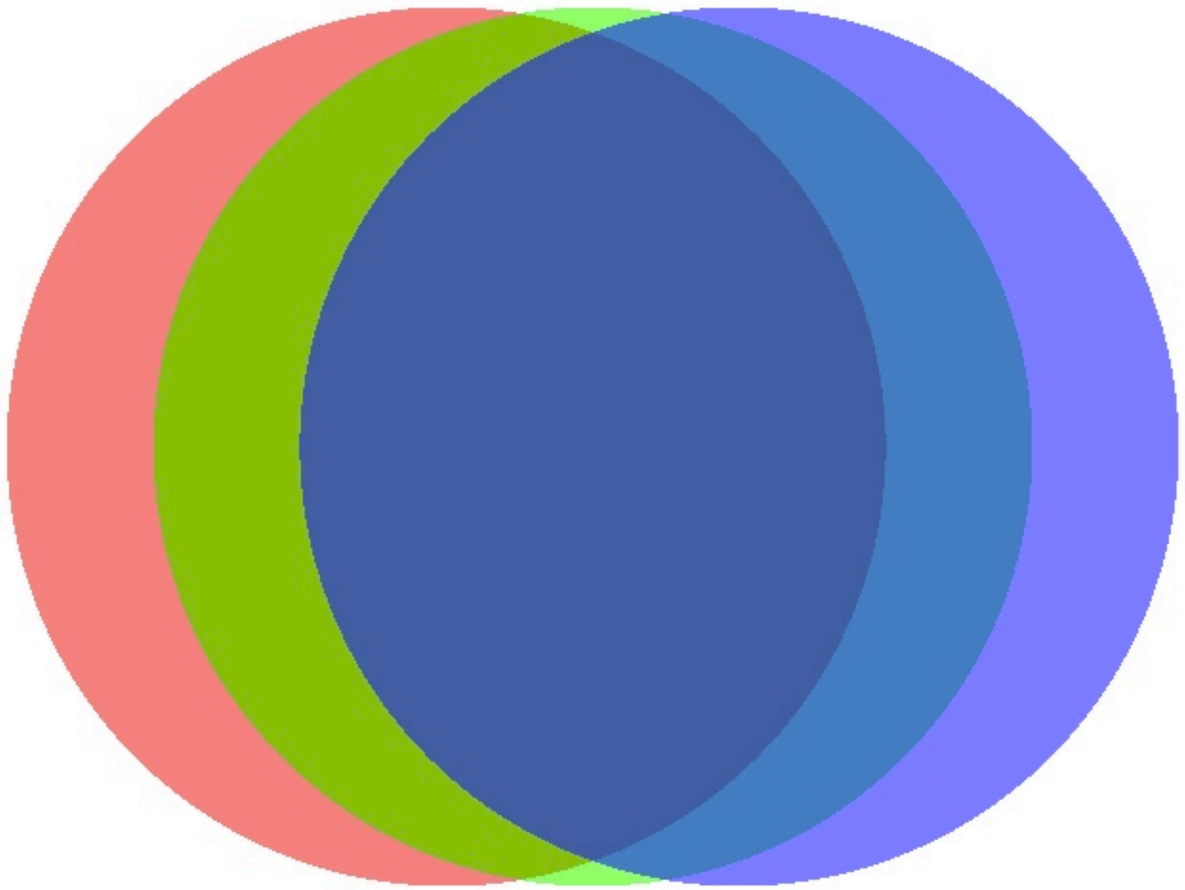
Atomicity and order

Requirements:

Image update must be atomic

Image updates must occur in proper order

(Note image update math is NOT commutative)



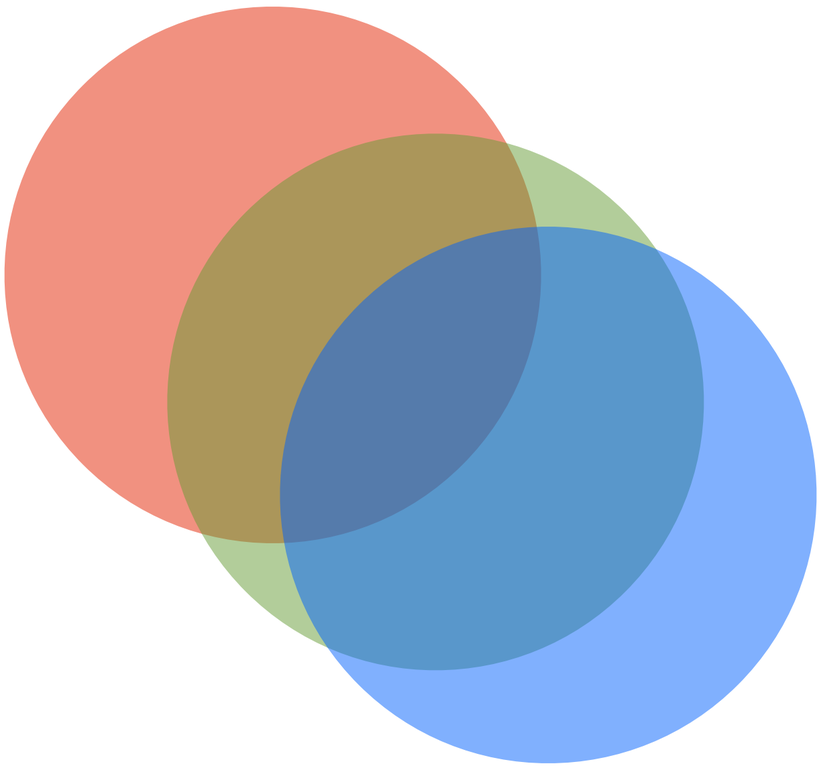
Example:

Red, green, and blue circles (each is 50% transparent: Common to store opacity "alpha")

Red circle is the farthest, blue closest

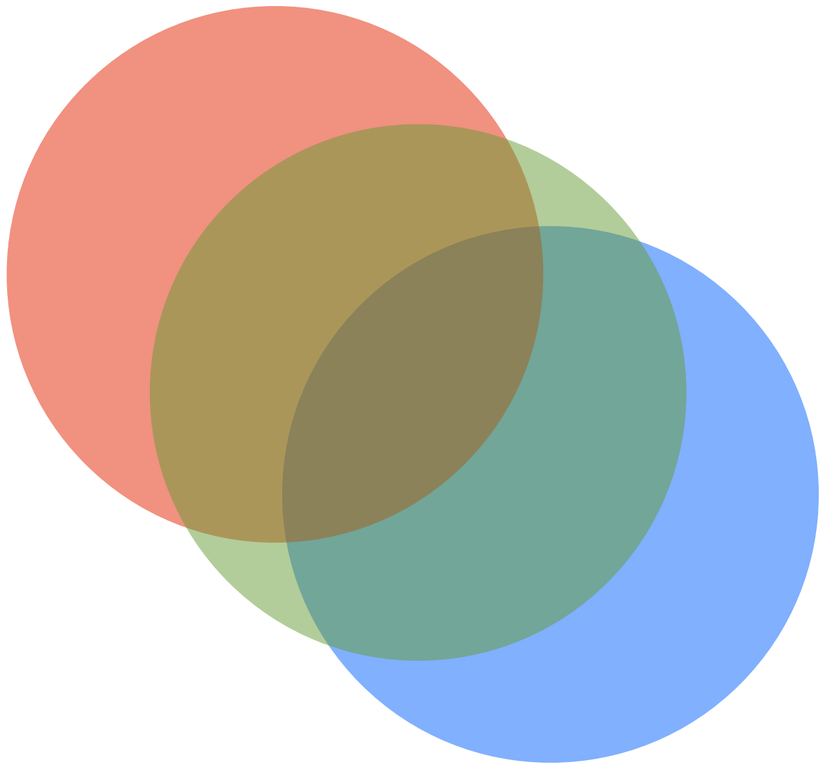
```
alpha_r = .5
alpha_g = .5
alpha_b = .5

image_color = alpha_r * red // contribution due to red
image_color = alpha_g * green + (1-alpha_g) * image_color // due to green
image_color = alpha_b * blue + (1-alpha_b) * image_color // due to blue
```

Order



Correct order:

blue over green over red



Incorrect order:

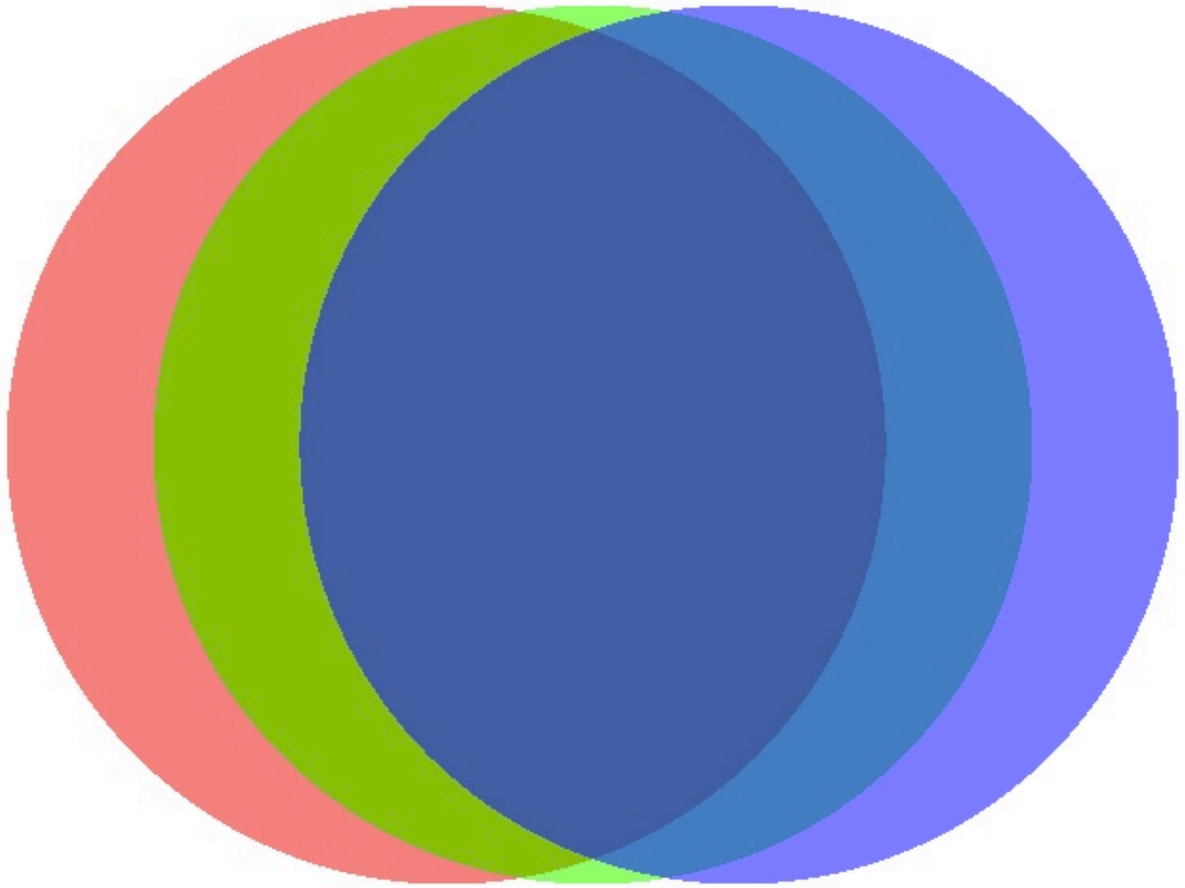
Atomicity and order

Requirements:

Image update must be atomic

Image updates must occur in proper order

(Note image update math is NOT commutative)



Proper order in this example is reverse depth order.

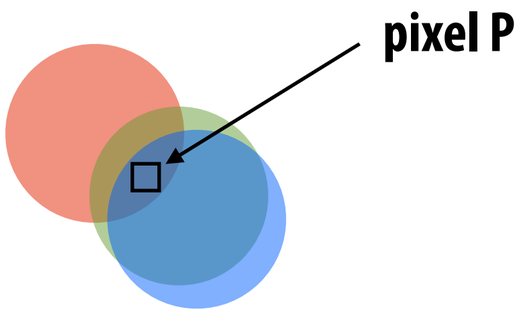
In this assignment we follow real time graphics systems and define order to be triangle order: ***for any pixel P, contribution of triangle 1 to P must be applied to image prior to contribution from triangle 2***

(we'll hand your renderer circles in reverse depth order)

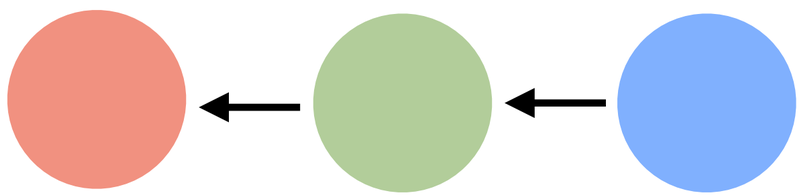
Order

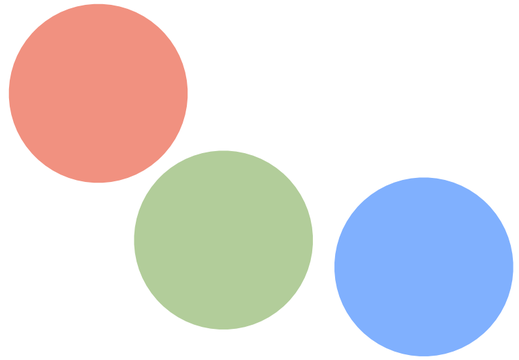
For any image pixel P, contribution of triangle 1 to P must be applied to image prior to contribution from triangle 2

Three input triangles



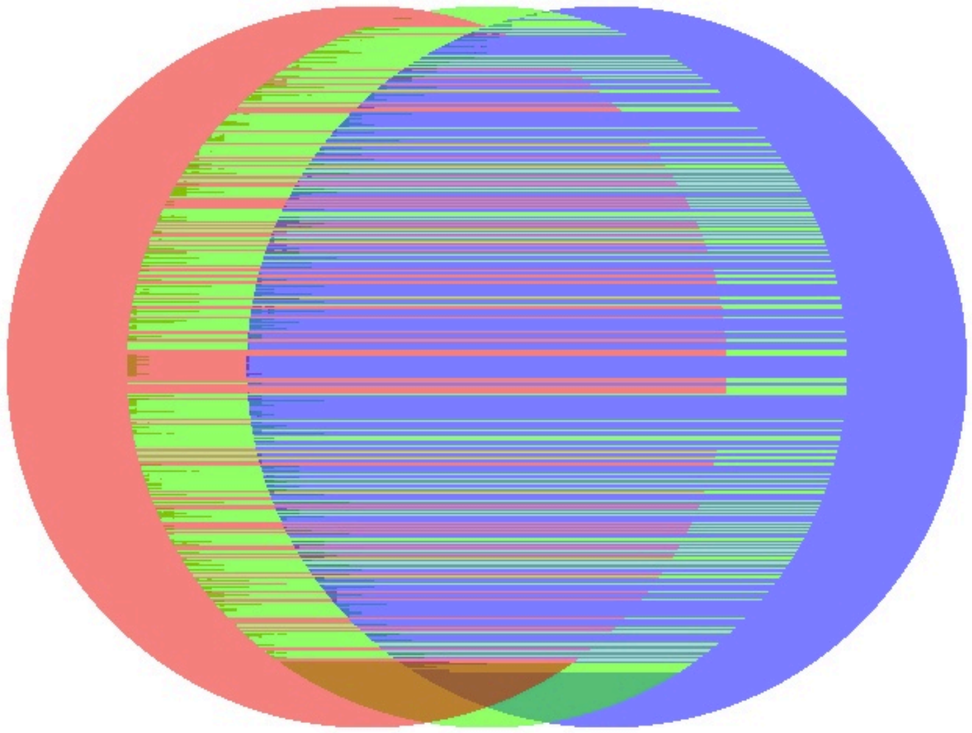
Dependencies between circles

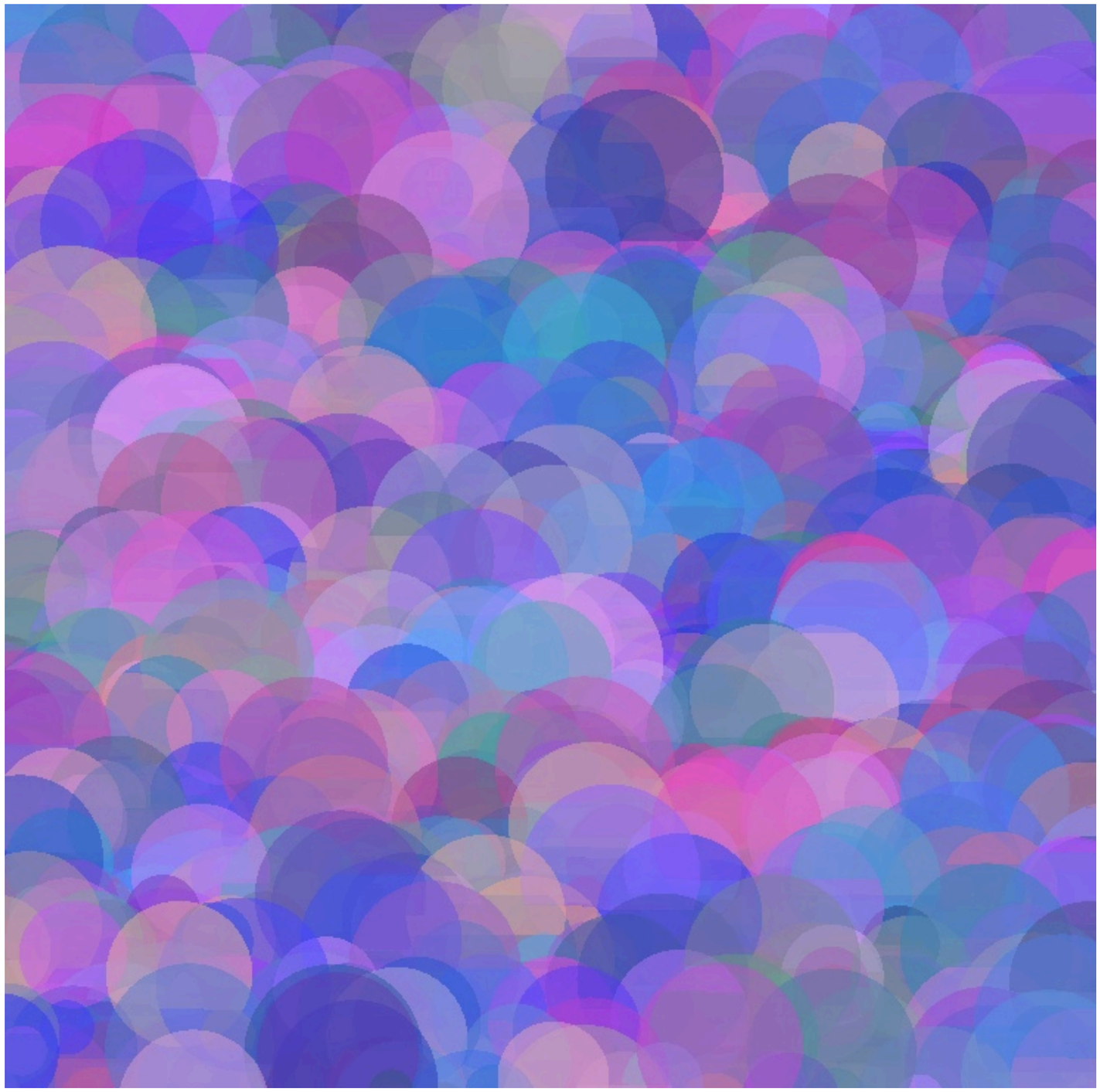




No overlap: no dependencies

Atomicity and ordering





Starter code:

1. **Fully functional sequential CPU renderer**
2. **GPU renderer in CUDA**
   - **Trivial parallelism over circles**
   - **Does not preserve order or atomicity**
   - **Results have artifacts**

These artifacts are non-deterministic (depend on computation schedule)

Assignment 2: implement the fastest, (correct) circle renderer you can

Today: more parallel program optimization

- Recall first lecture: distributing work to processors
  - Goal: minimizing overhead while also achieving a workload balance
  - Static vs. dynamic assignment, several work queue implementations
  - Keep it simple, stupid (profile, analyze, then tune if required)
- Today: minimizing communication and exploiting locality

Terminology

Latency

The amount of time needed for an operation to complete.

Example: A memory load that misses the cache has a latency of 200 cycles.

A packet takes 20 ms to be sent from my computer to Google.

Bandwidth

The rate at which operations are performed.

Example: Memory can provide data to the processor at 25 GB/sec.

A communication link can send 10 million messages per second.

Cost

The (detrimental) effect operations have on program execution time (or some other metric, e.g, power)

"My slow program sends most of it's time waiting on memory." (cost of latency)

"Saxpy achieves low ALU utilization because it is bandwidth bound." (cost of insufficient bandwidth)

A very simple model of communication

$$T(n) = T_0 + \frac{n}{B}$$

T = transfer time (overall latency of the operation)

T_0 = start-up cost

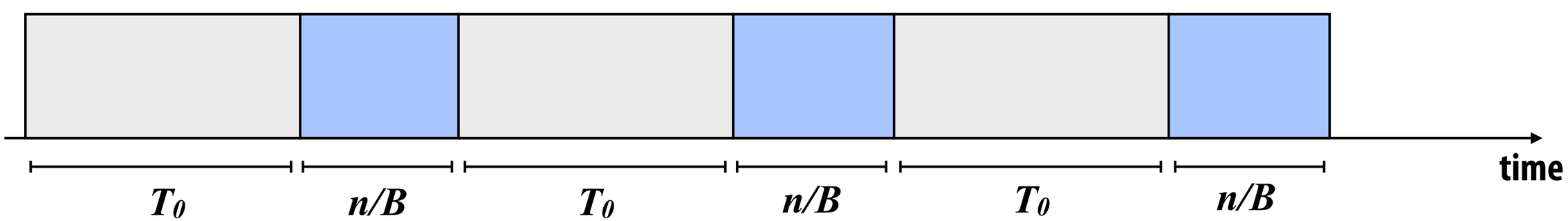
n = bytes transferred

B = transfer rate (bandwidth of the link)

Assumption: processor does no other work while waiting for transfer to complete ...

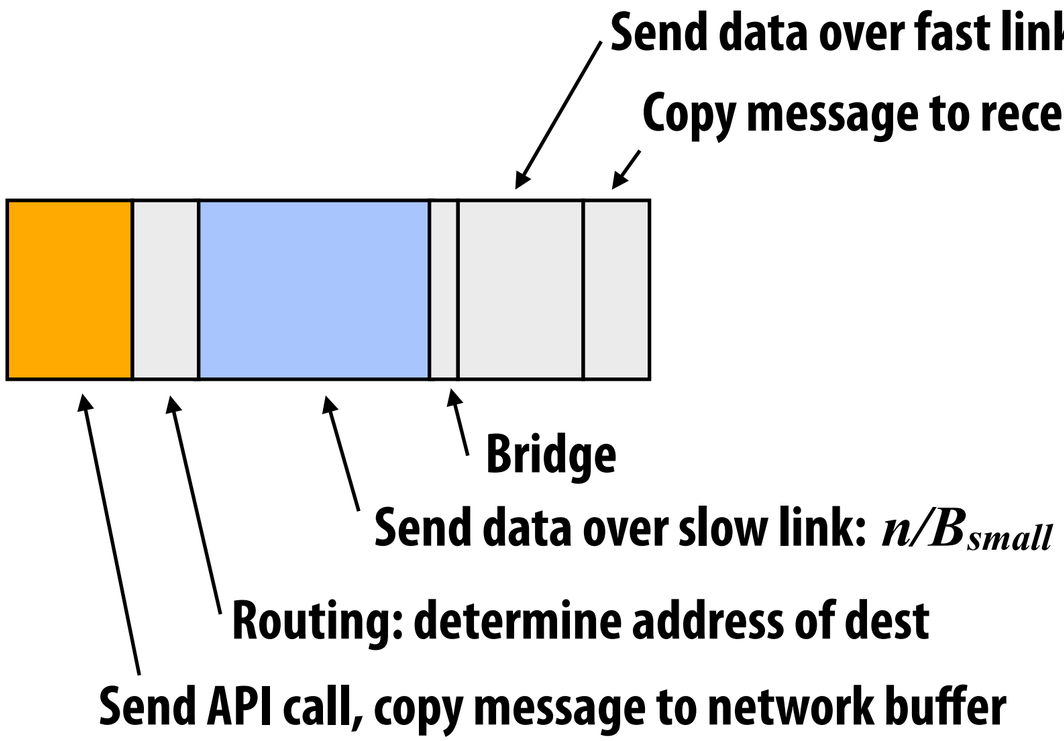
Effective bandwidth = $n / T(n)$

Effective bandwidth depends on transfer size



A more general model of communication

Communication time = overhead + occupancy + network delay



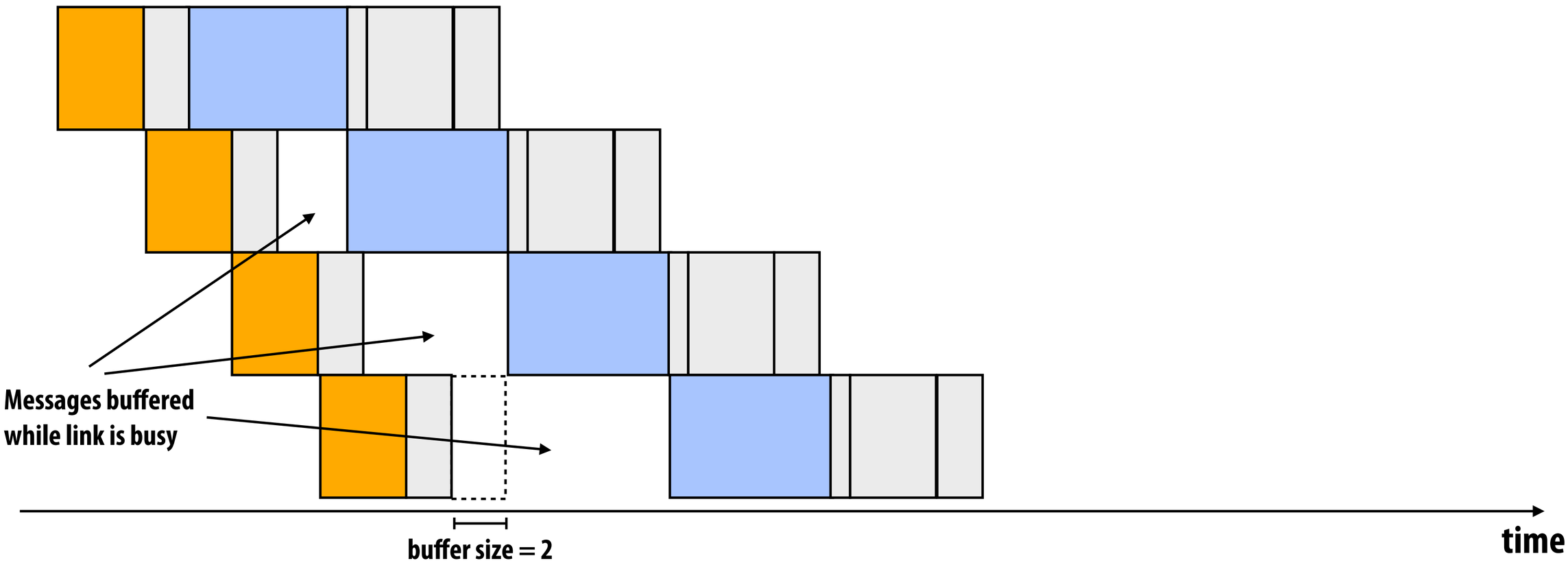
Example: sending a message

= Overhead (time spent on the communication by a processor)

= Occupancy (time for data to pass through slowest component of system)

= Network delay (everything else)

Pipelined communication



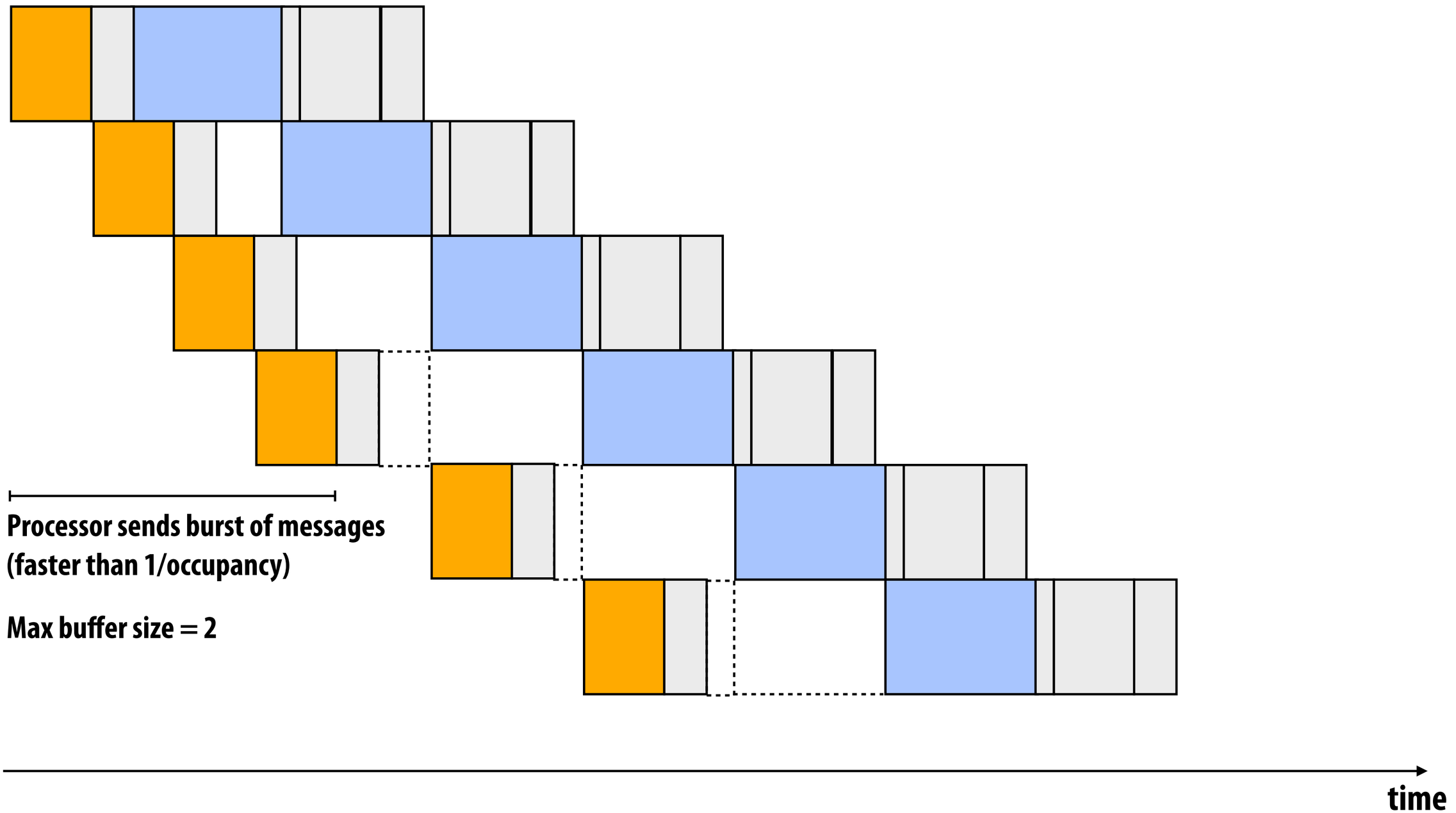
Occupancy determines communication rate (effective bandwidth)

= Overhead (time spent on the communication by a processor)

= Occupancy (time for data to pass through slowest component of system)

= Network delay (everything else)

Pipelined communication



Occupancy determines communication rate

(in steady state: msg/sec = 1/occupancy)

Communication cost

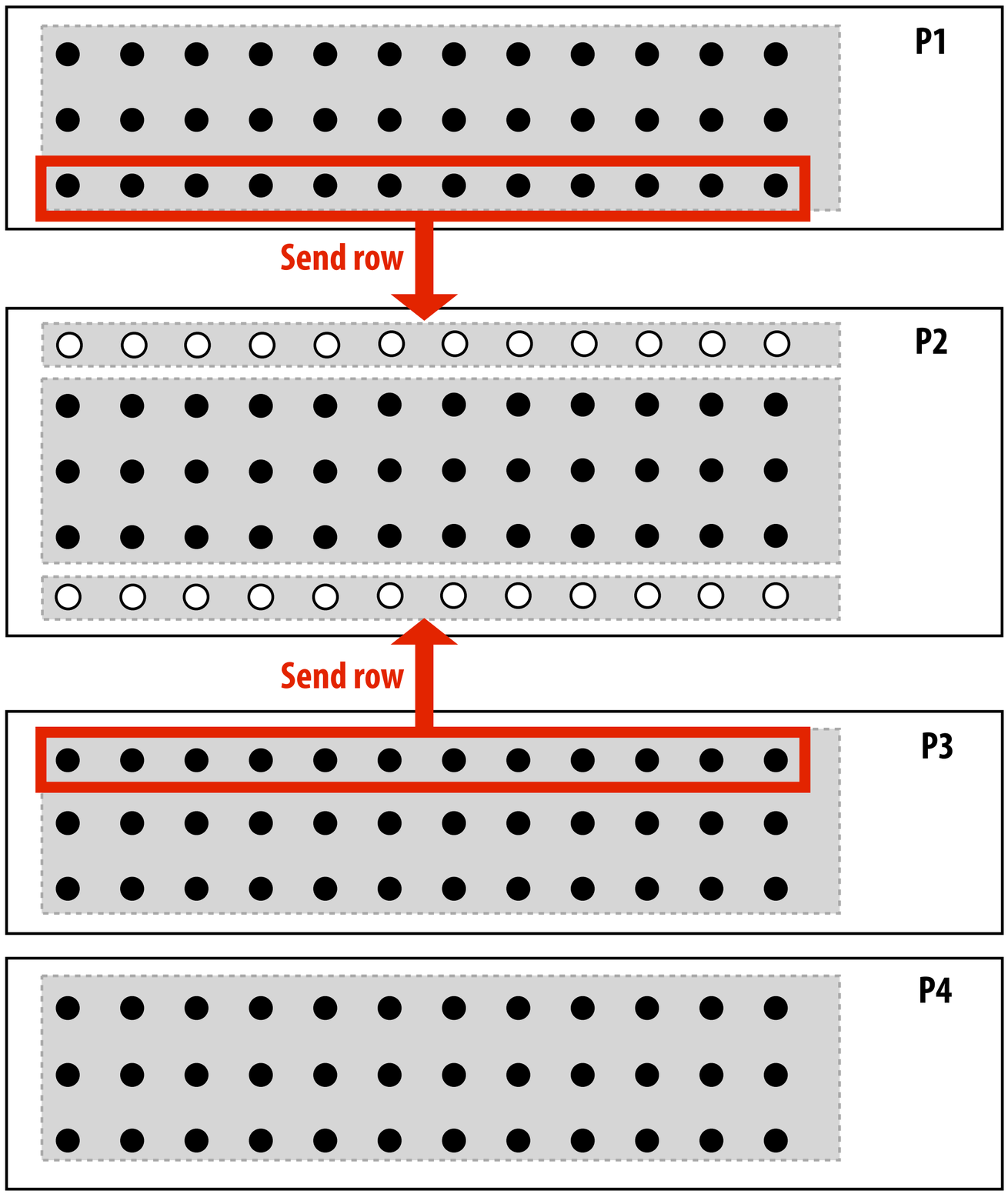
Total communication cost = frequency x (communication time - overlap)

Overlap: portion of communication performed concurrently with other work

"Other work" can be computation or other communication (as in the previous example)

Remember, what really matters is not the absolute cost of communication, but it's cost relative to the cost of the computation fundamental to the problem being solved.

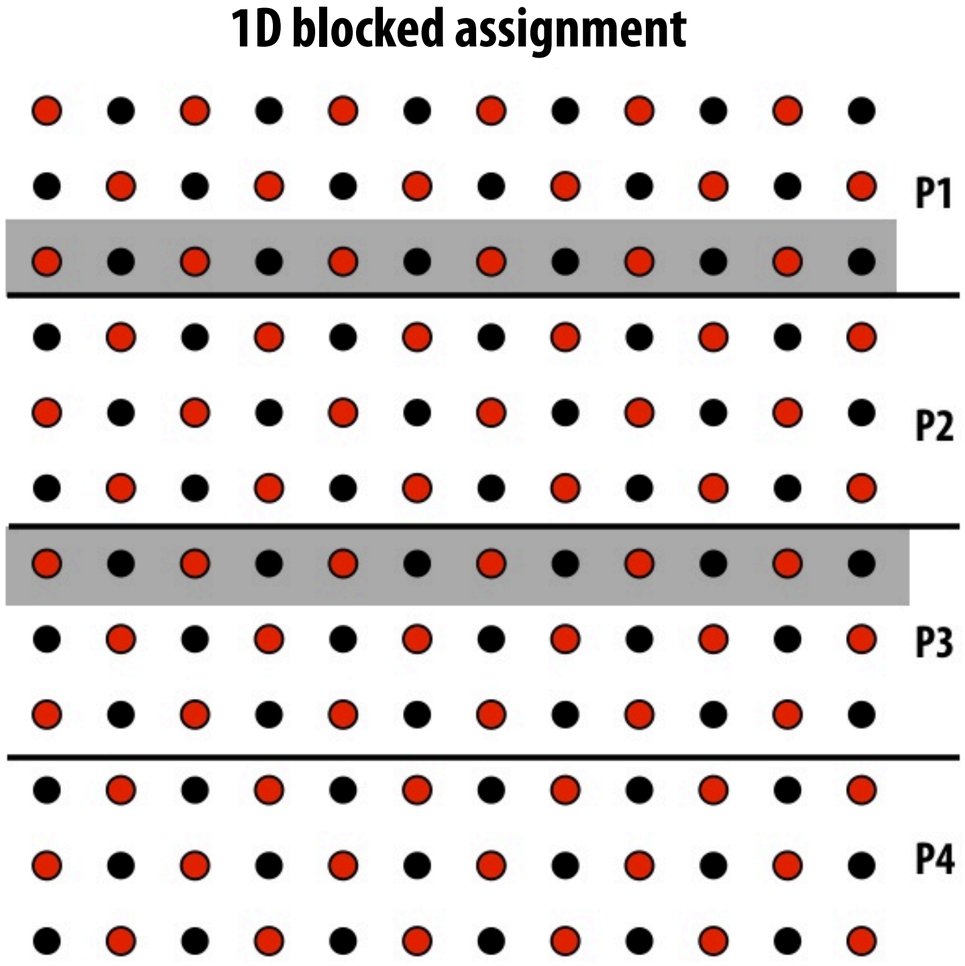
Inherent communication



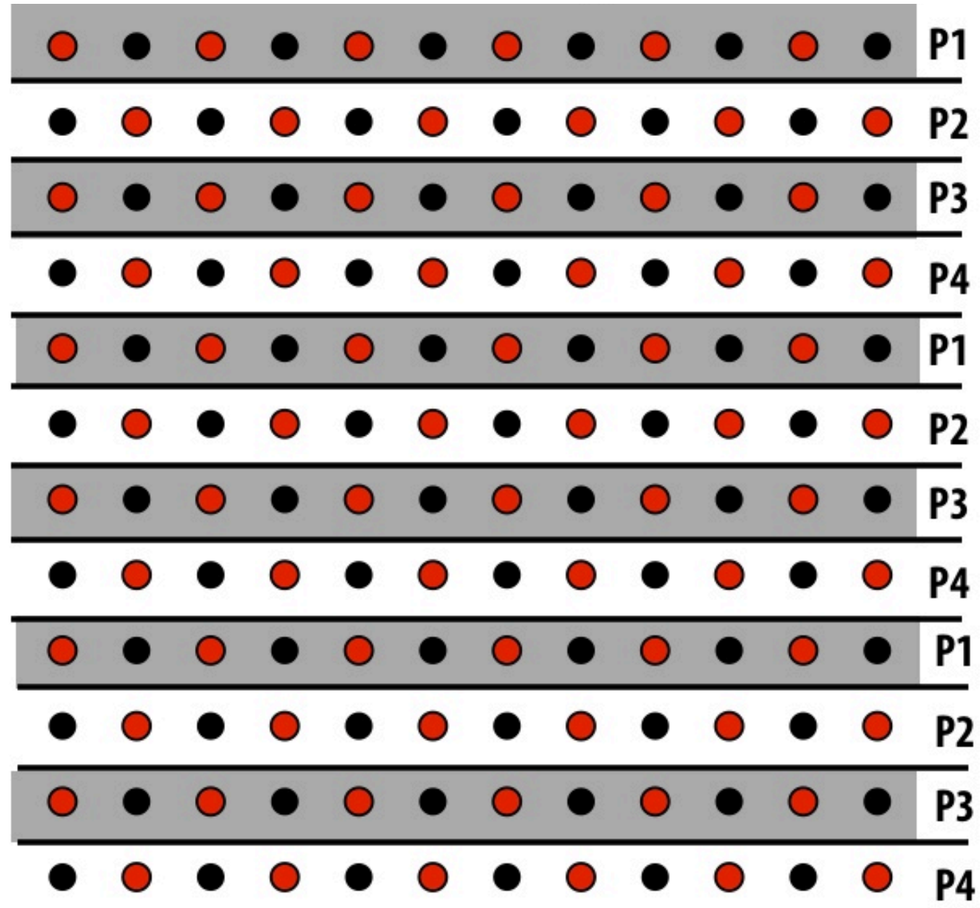
Communication that must occur in a parallel algorithm. Fundamental to the algorithm.

Reducing inherent communication

- Good assignment can reduce inherent communication (decrease communication to computation ratio)



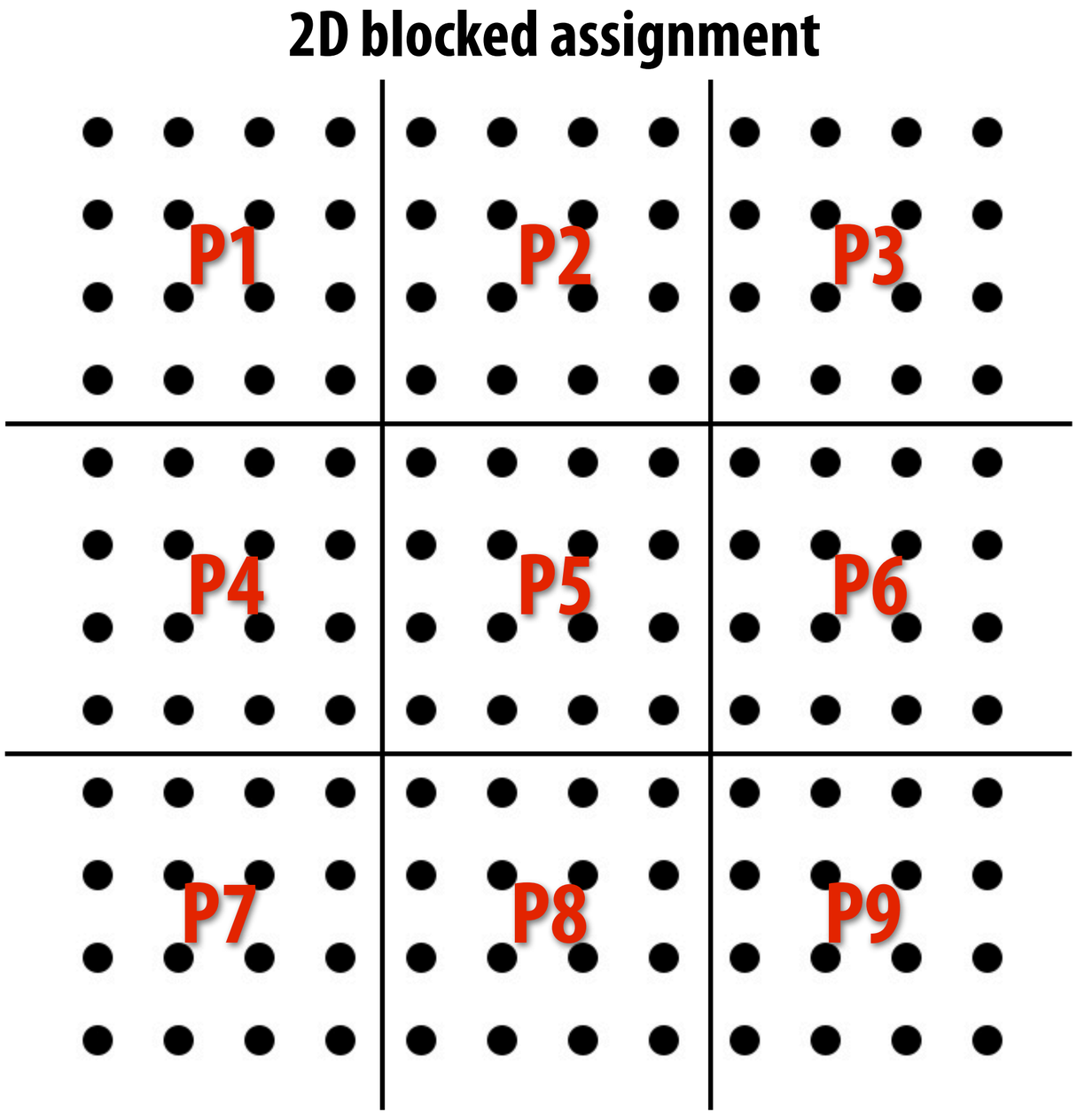
$$\frac{\text{elements communicated}}{\text{elements computed}} = 4/3$$



$$\frac{\text{elements communicated}}{\text{elements computed}} = 4/1$$

(each phase: send two rows, recv two rows)

Reducing inherent communication



N^2 elements

P processors

elements computed: $\frac{N^2}{P}$

elements communicated: $\propto \frac{N}{\sqrt{P}}$

comm-to-comp ratio: $\frac{\sqrt{P}}{N}$

Asymptotically better communication scaling than 1D blocked assignment

Communication costs increase sub-linearly with P

Assignment captures 2D locality of algorithm

Communication-to-computation ratio

$$\frac{\text{amount of communication}}{\text{amount of computation}}$$

- If denominator is execution time of computation, ratio gives average bandwidth requirements

- Another term: "arithmetic intensity" = 1 / communication-to-computation ratio

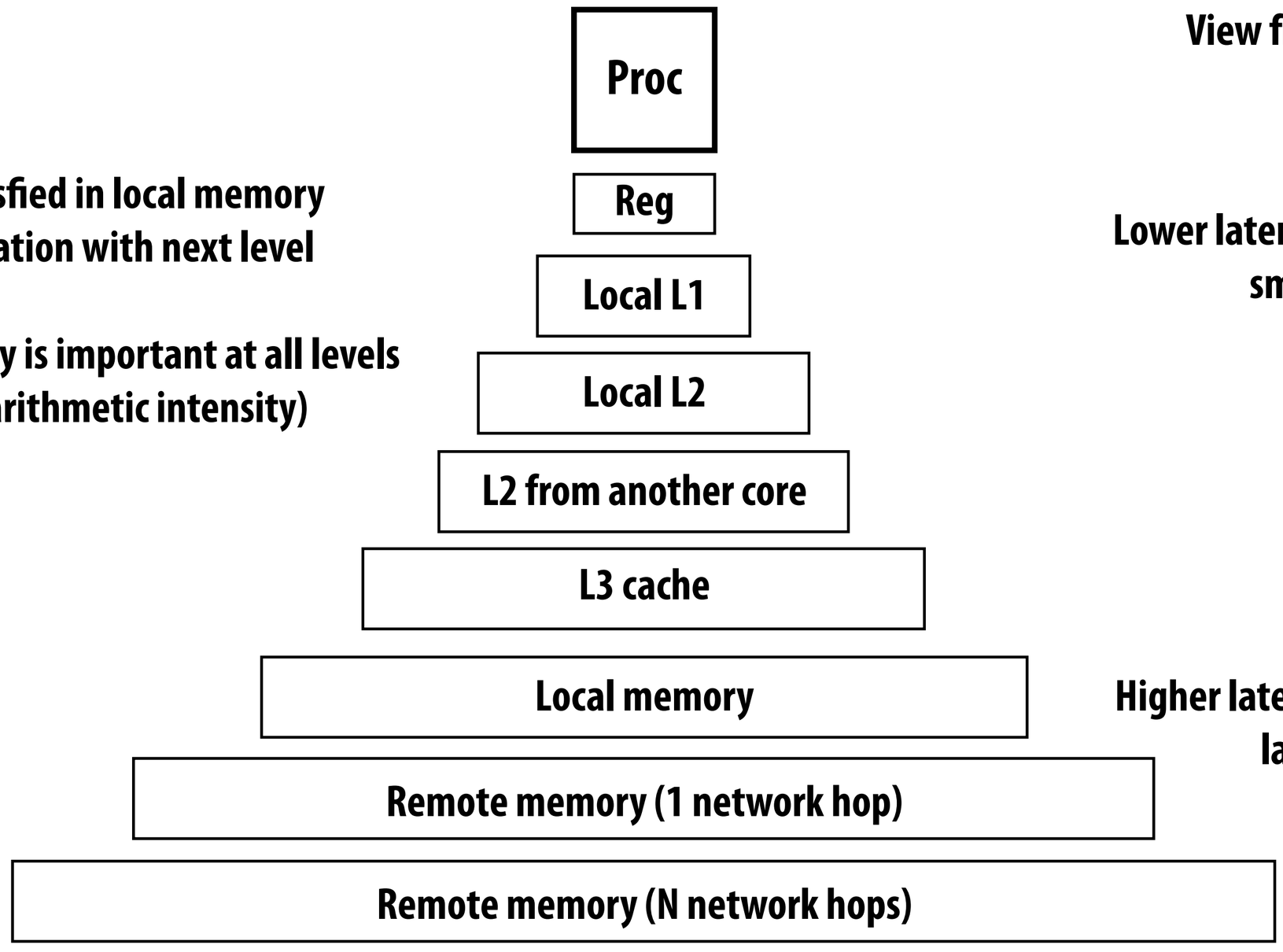
- High arithmetic intensity needed on parallel processors, since the ratio of compute capability to available bandwidth is very high (recall SAXPY)

Parallel system as an extended mem hierarchy

- Up until now: adopted notion that data was partitioned amongst processors, and that "non-local data" required communication
- In reality, parallel system is multi-memory, multi-cache system. Characteristics of data access latencies and bandwidths can have large effect on performance

Accesses not satisfied in local memory cause communication with next level

Managing locality is important at all levels (achieving high arithmetic intensity)



View from one processor

Lower latency, higher bandwidth, smaller capacity

Higher latency, lower bandwidth, larger capacity

Artifactual communication

- Inherent communication: assumes unlimited capacity, small transfers, perfect knowledge of what is needed to communicate

- Artifactual communication is everything else (depends on interaction of application and system)
  - System-granularity block transfers (send more then what's needed: e.g., read one word, load entire cache line)
  - Poor allocation of data among distributed memories
  - Finite replication capacity (in any level of hierarchy)

Review of the three (now four) Cs

- **Cold miss**

  First time data touched. Unavoidable.

- **Capacity miss**

  Working set larger than cache. Can be decreased by larger caches

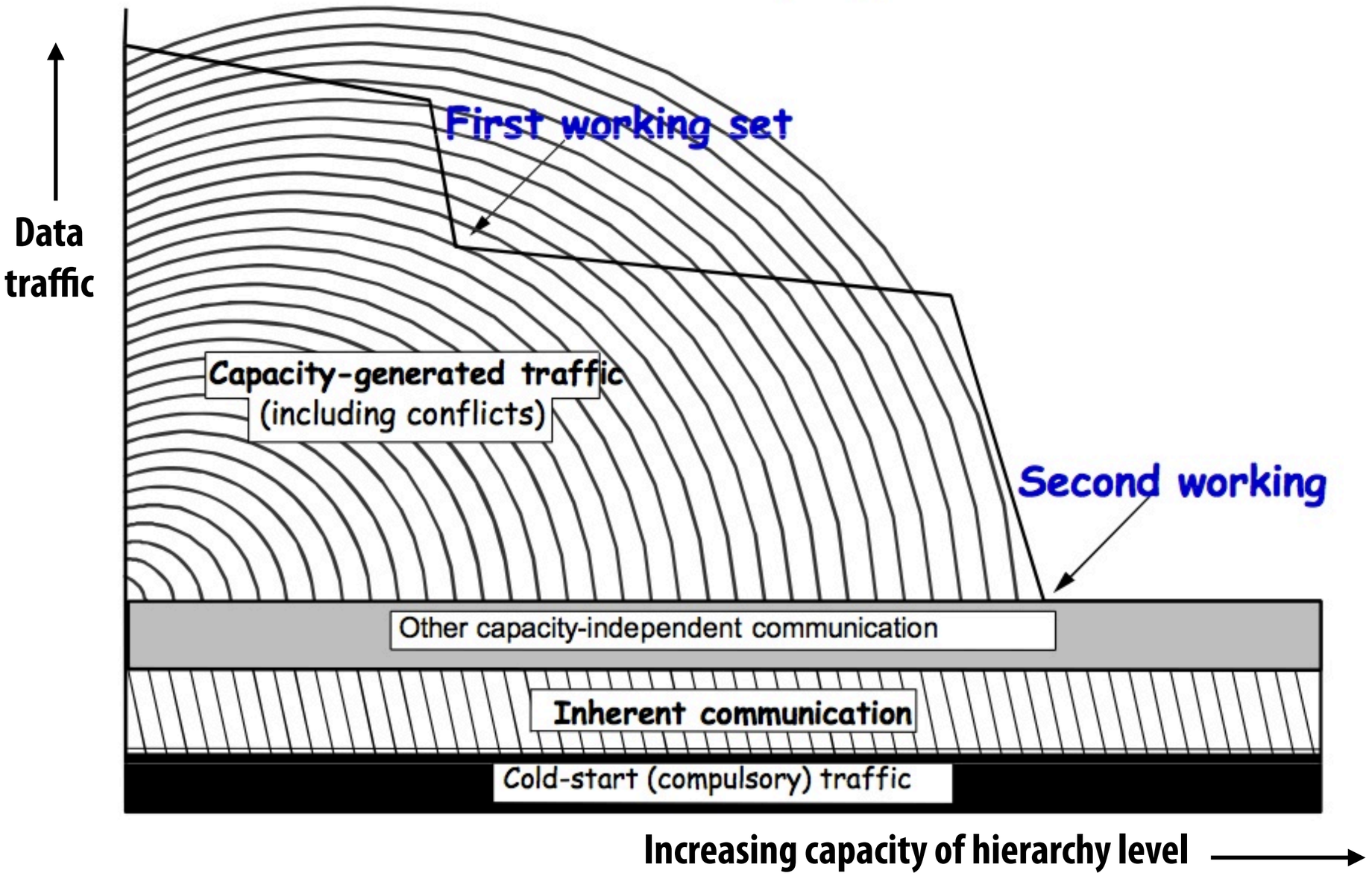
- **Conflict miss**

  Miss induced by cache management policy. Can reduce by changing cache associativity, or data access pattern in application

- **Communication miss (new)**

  Due to inherent communication or artifactual communication in parallel system

Working set perspective



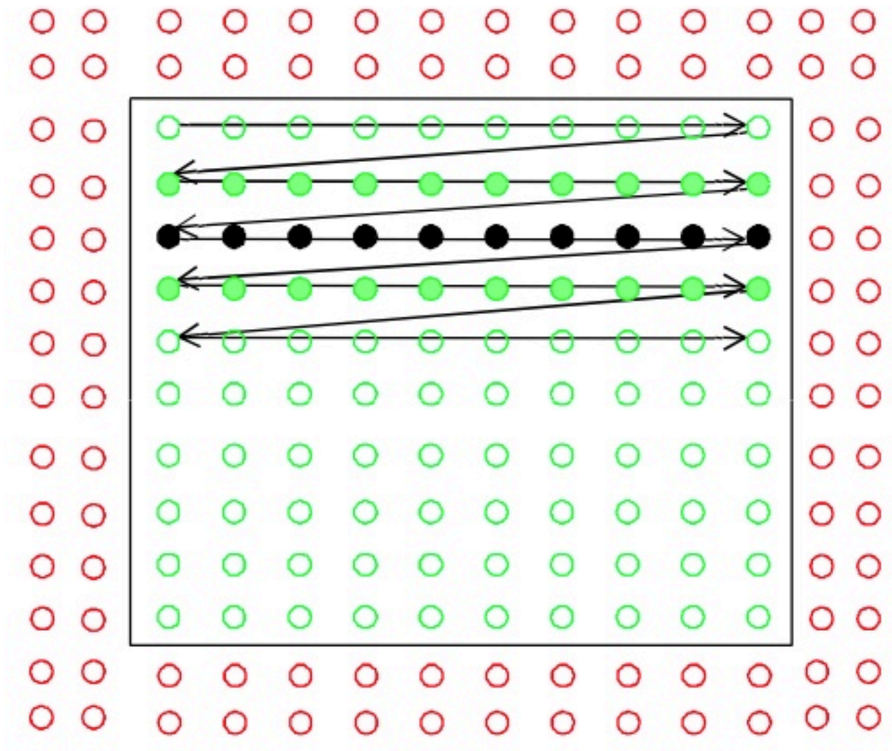
This diagram holds true at any level of the memory hierarchy in a parallel system

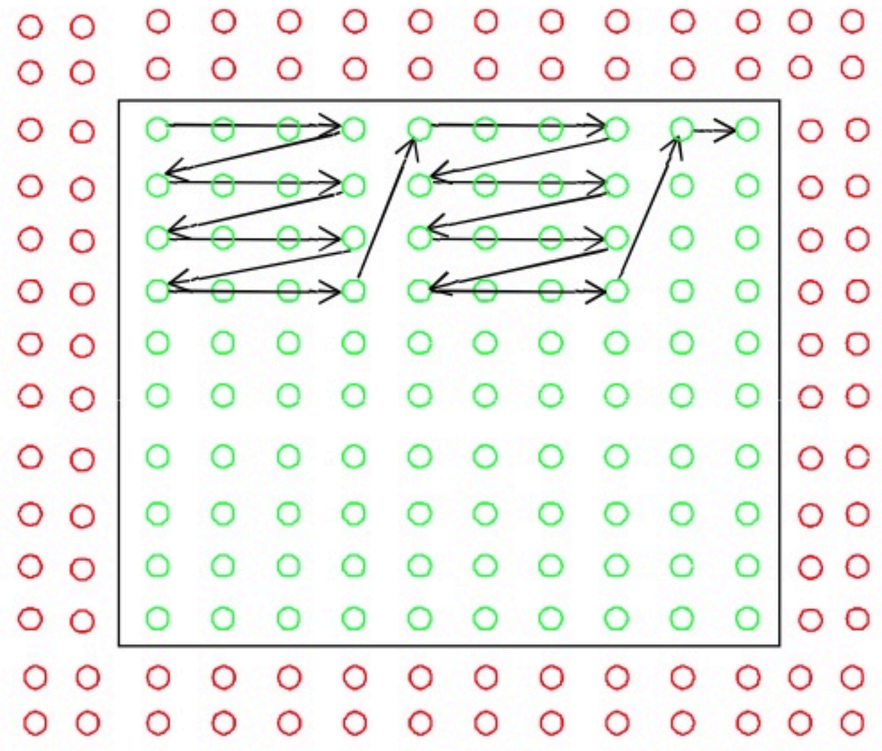
Question: how much capacity should an architect build for this workload?

Reducing amount of communication

Improve temporal locality

- "Blocking": reorder computation to make working sets map well to system's memory hierarchy





Recall matrix transpose assignment in 15-213:
Main idea: replicate block of data in local memories (cache)
Process it in its entirety (accessing many times) prior to moving only next block

Tip: how might blocking apply in a renderer?

Improve temporal locality

- Exploit sharing: co-locate tasks that operate on the same data
  - Schedule threads working on the same data structure at the same time on the same processor
  - Reduces inherent communication
- Example: CUDA thread block
  - Abstraction to localize related processing in the machine
  - Threads in block often cooperate to perform an operation
  - Leverage fast access to / synchronization via CUDA shared memory

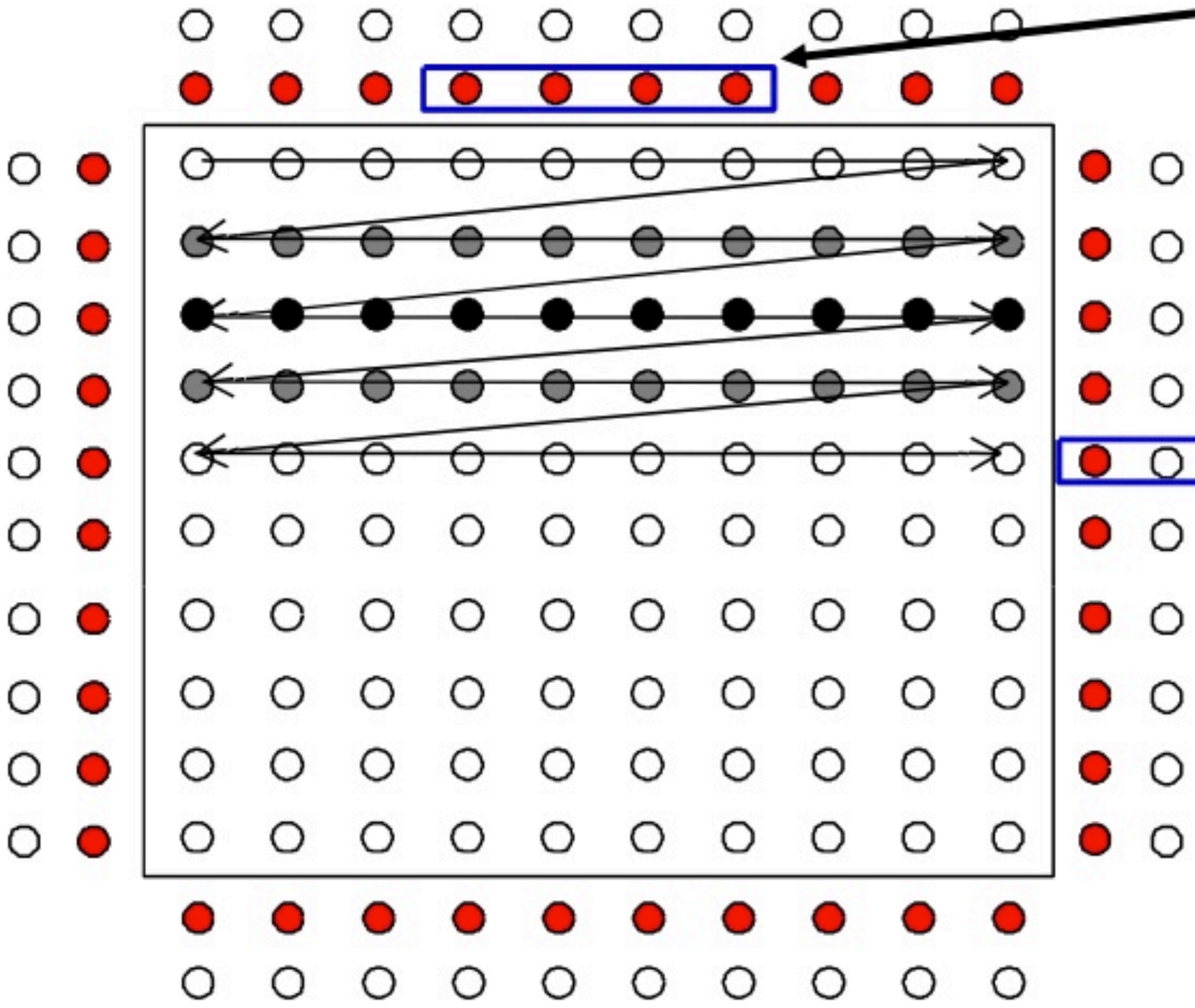
Exploiting spatial locality

- Granularities can be very important
  - Granularity of allocation
  - Granularity of communication / data transfer
  - Granularity of coherence (future lecture)

Artifactual communication due to comm. granularity

Shared memory system. Cache line communication granularity (four elements)

2D blocked partitioning



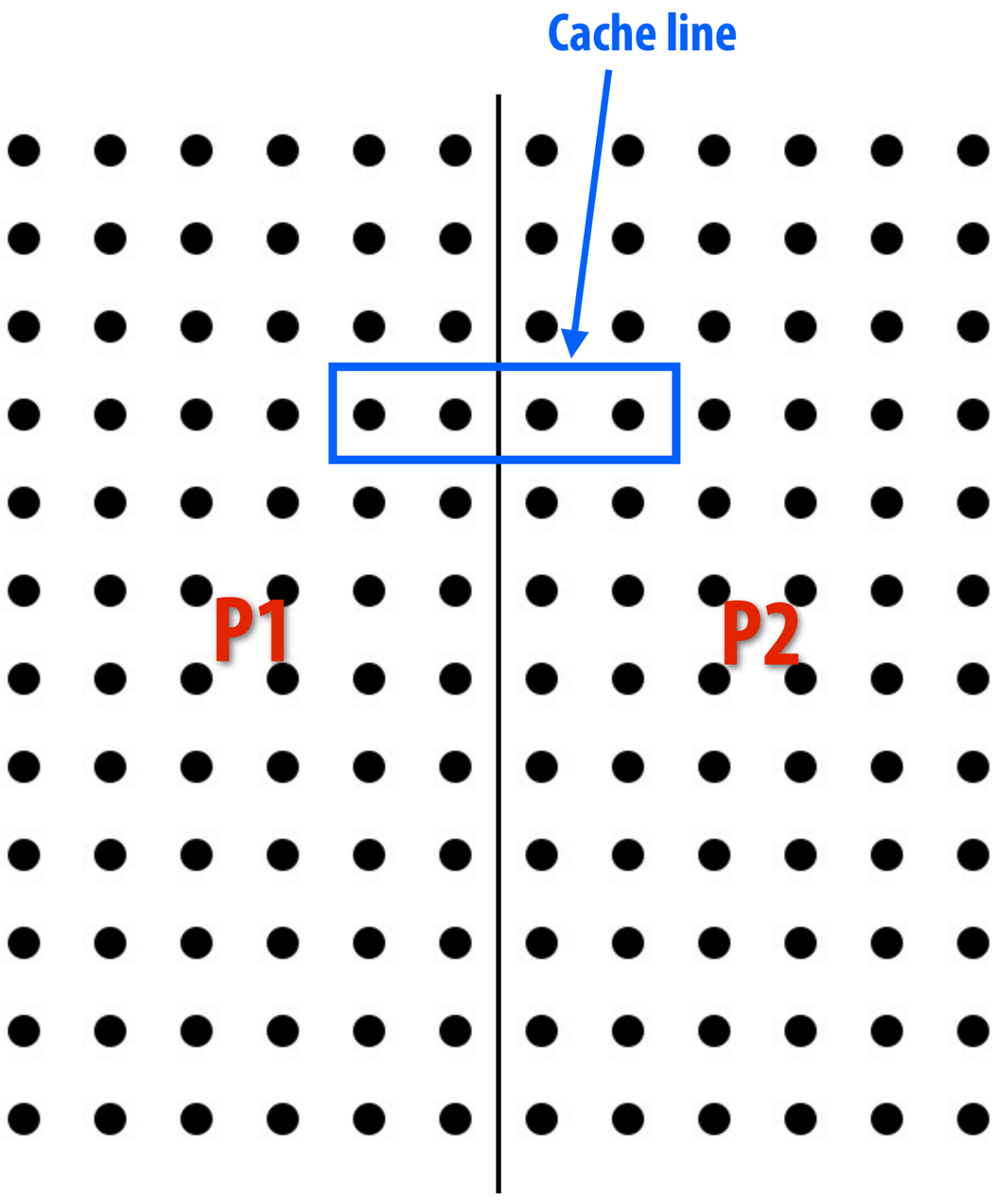
Good spatial locality for non-local accesses to top-bottom rows

Poor spatial locality for non-local accesses to left-right columns

Need one element from line. Must communicate four.

Implication: 1D blocked layout may perform better despite worse inherent communication-to-computation ratio.

Artifactual comm. due to comm./coherence granularity



Data partitioned among memories local to processors

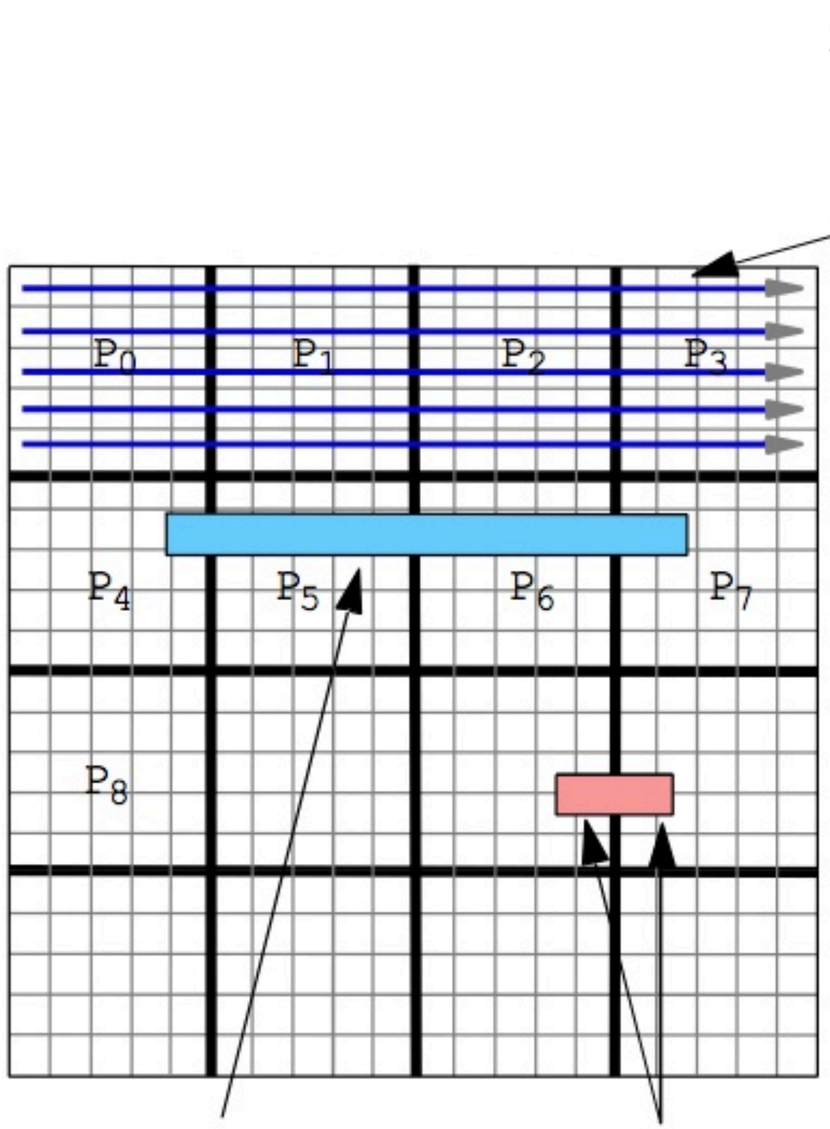
Processors access their assigned elements (no inherent communication)

But data access on real machine triggers communication (artifactual)

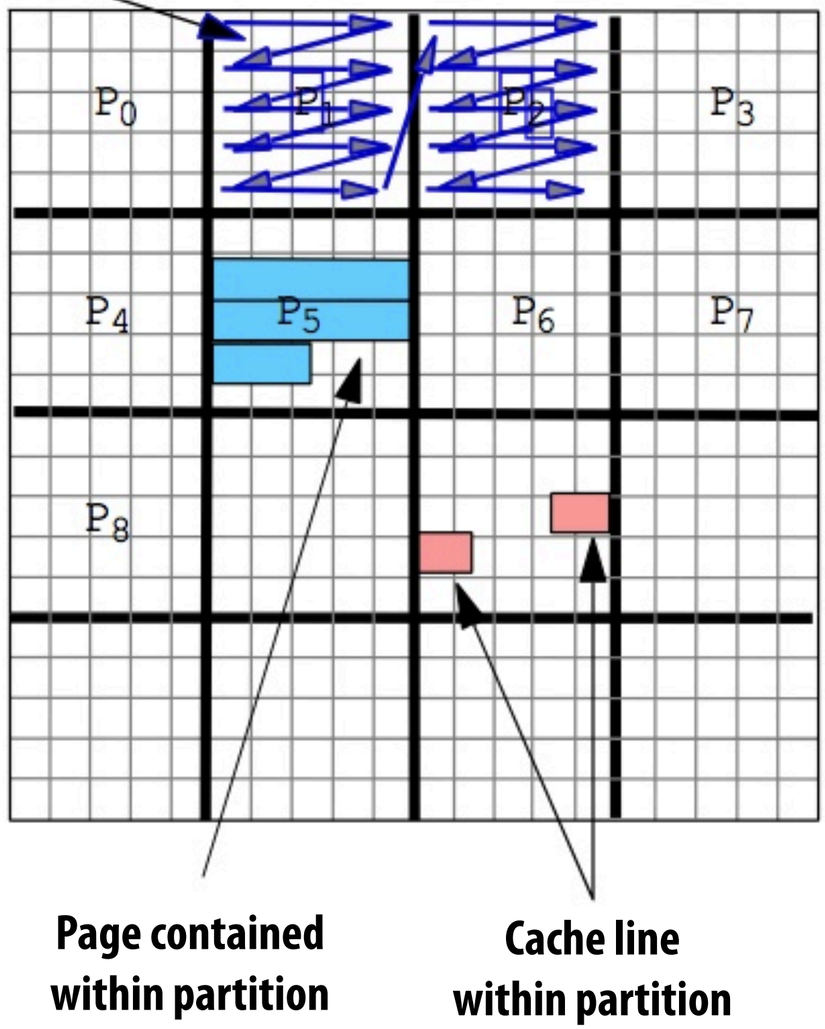
Also, writes by different processors require cache coherence **

** Implementing cache coherence will be a topic of future lectures.

Reducing artifactual comm: better layout



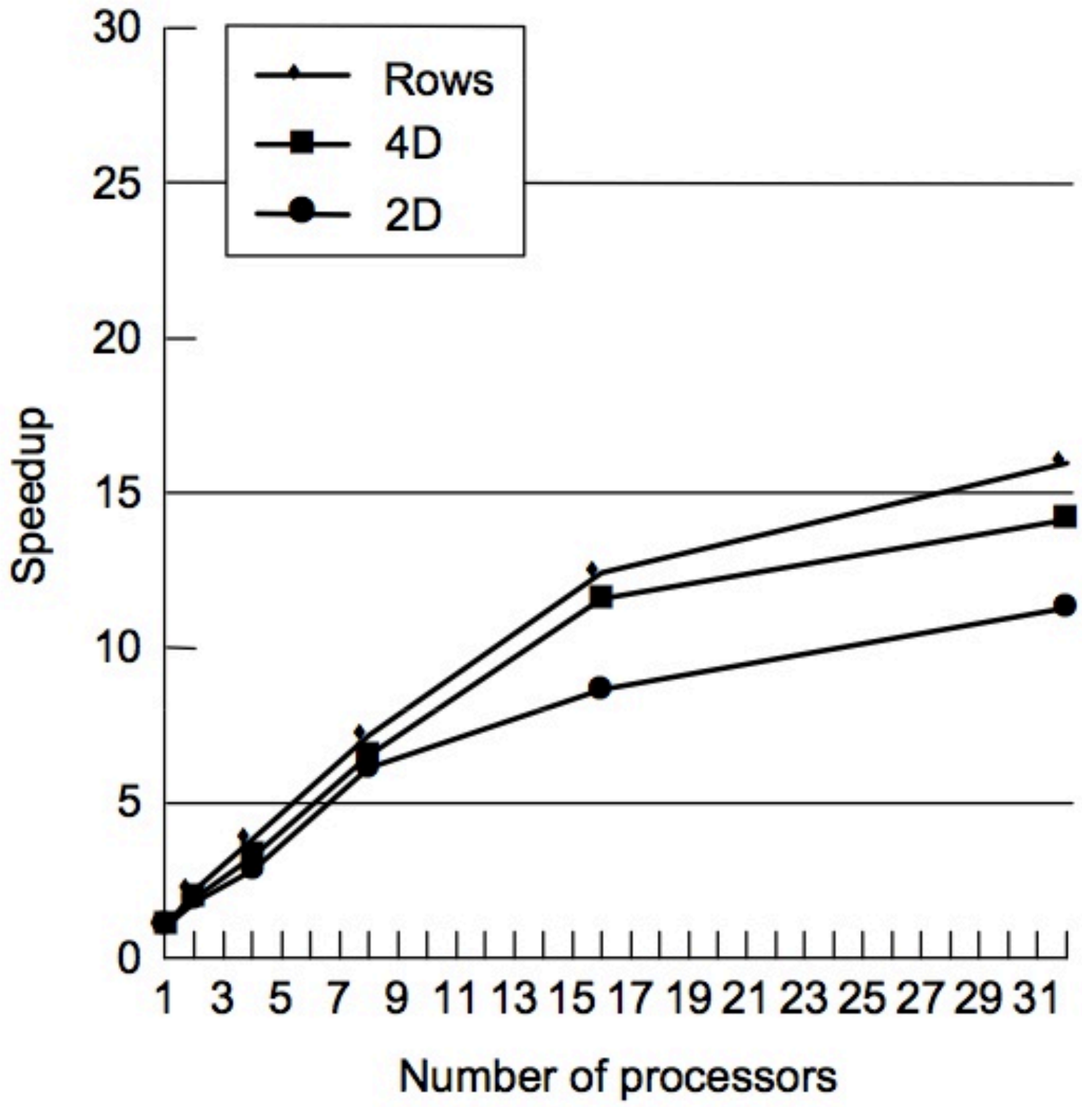
2D, row-major array layout



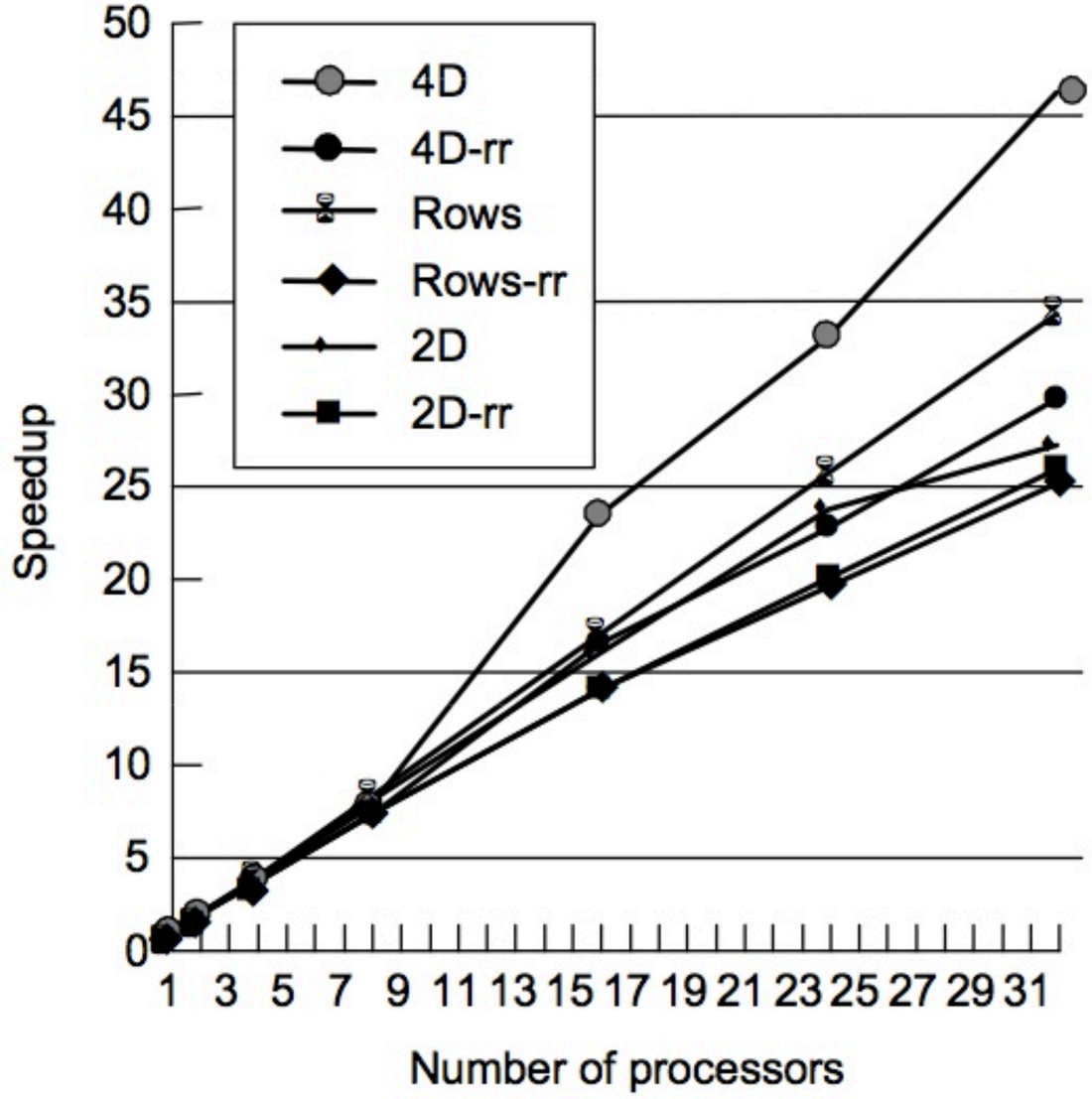
4D array layout (block-major)

Performance impact of 4D layout

(on SGI Origin 2000 shared address space machine)



Ocean (514x514 grids)



Solver kernel (12K x 12K grid)

Structuring communication to reduce cost

Total communication cost = frequency x (communication time - overlap)

Total communication cost = frequency x (overhead + occupancy + network delay - overlap)

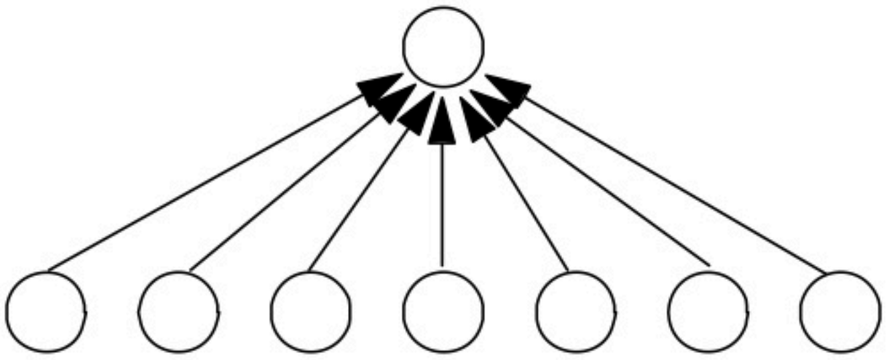
Total communication cost = frequency x (overhead + (n/B + contention) + network delay - overlap)

Here: occupancy dominated by time it takes to transfer message (n bytes) over slow link + delays due to contention for link

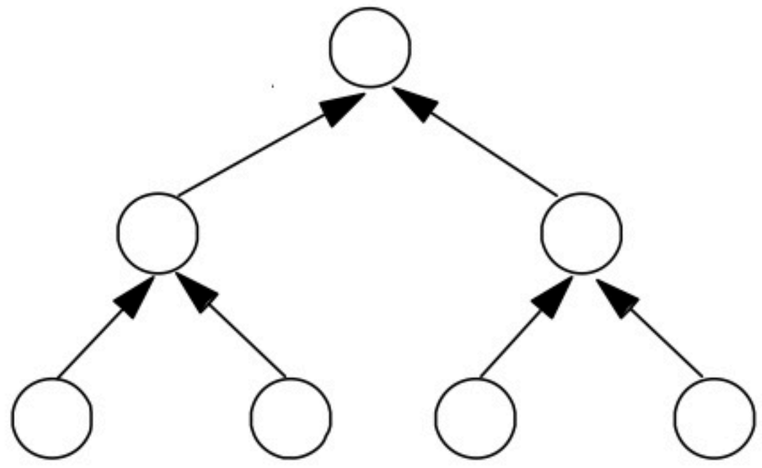
Contention

- All resources have non-zero occupancy
  - Memory, communication links, comm. controller. etc.
  - Each has fixed number of transactions per unit time
- Contention occurs when many requests to a resource are made within a small window of time
  - Resource is a "hot spot"

Example: updating a shared variable



Flat communication:
potential for high contention

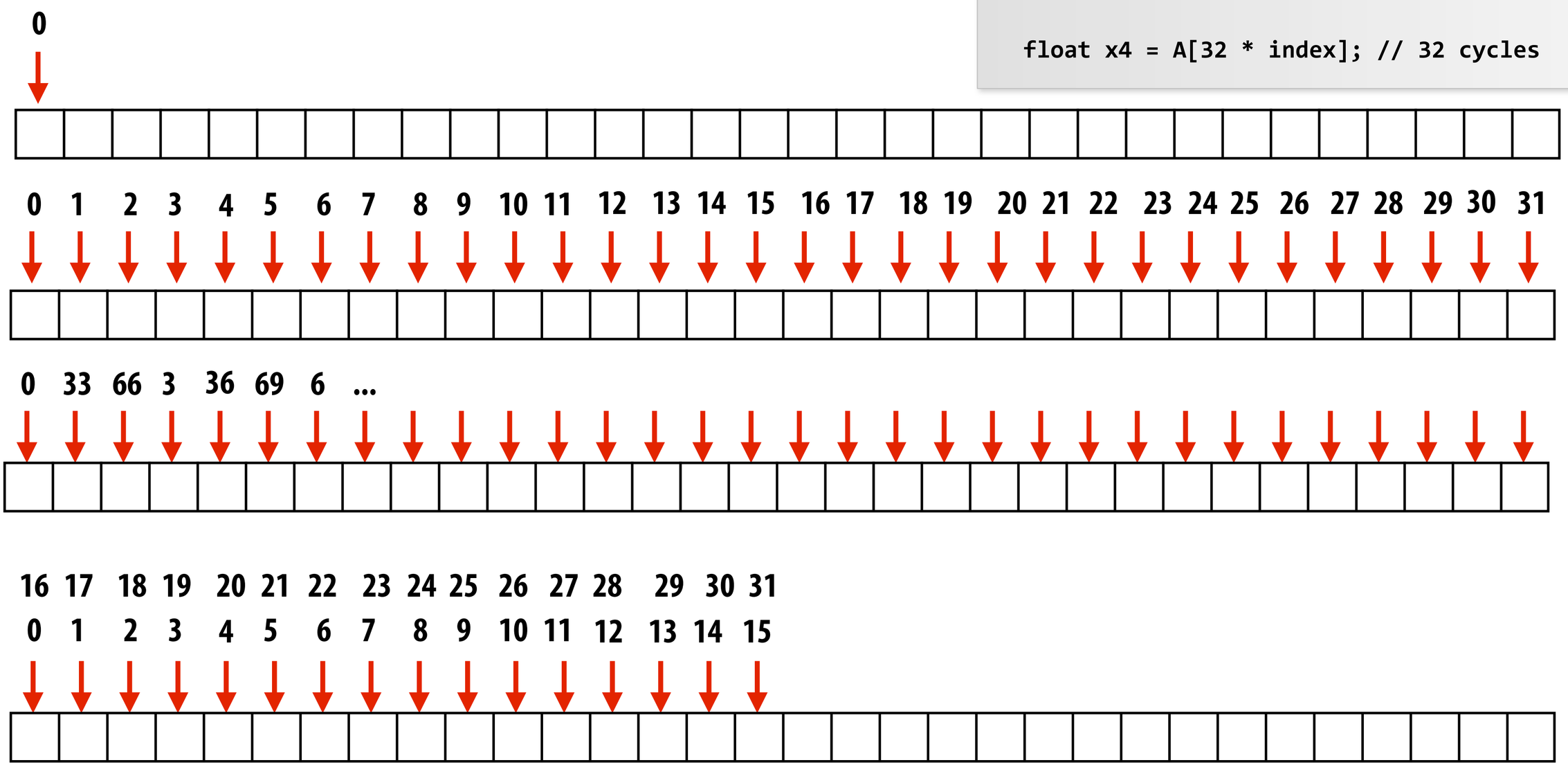


Tree structured communication:
reduces contention

NVIDIA GTX 480 contention example

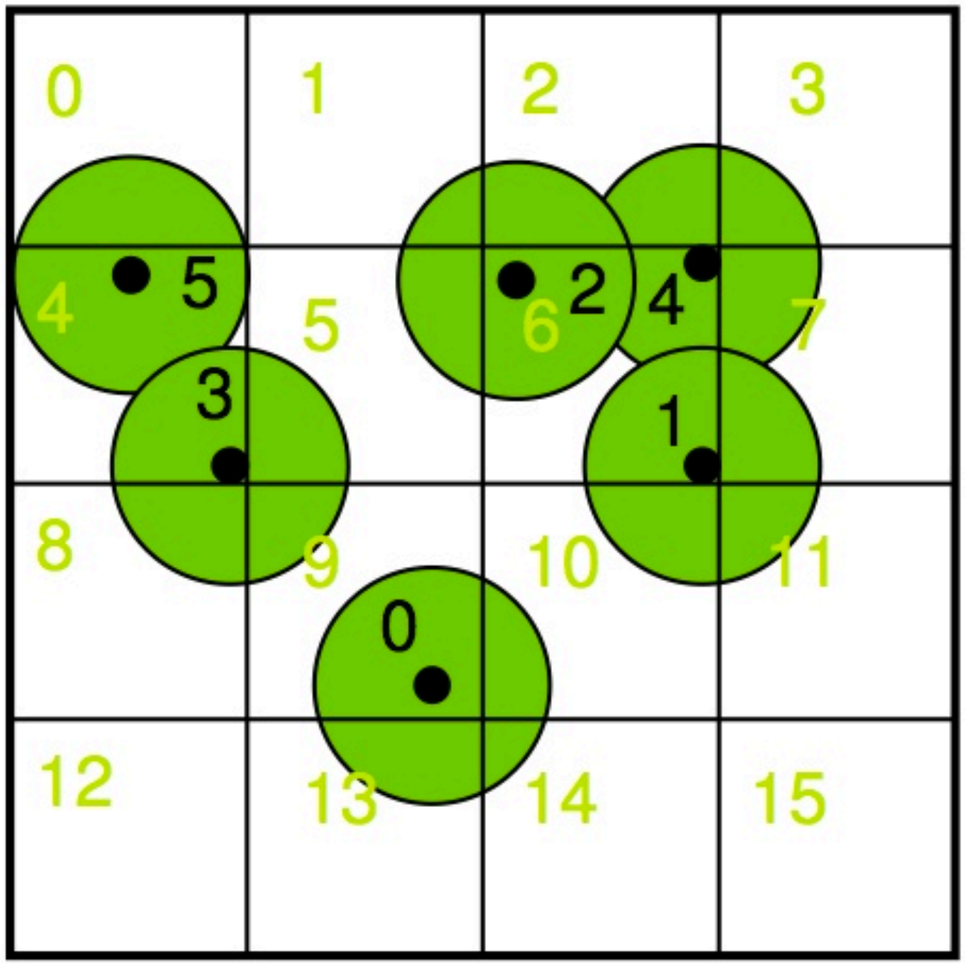
- Shared memory: 32 banks

```
__shared__ float A[512];


int index = threadIdx.x;
float x1 = A[0];  // single cycle

float x1 = A[index];  // single cycle

float x2 = A[3*index]; // single cycle

float x3 = A[index / 2];  // 2 cycles

float x4 = A[32 * index]; // 32 cycles
```



Another contention example

- 15 cores, up to 1024 threads per core
- Place a bunch of point particles in a 16 cell uniform grid



Cell id	Count	Particle id
0	0	
1	0	
2	0	
3	0	
4	2	3, 5
5	0	
6	3	1, 2, 4
7	0	
8	0	
9	1	0
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	

Reducing communication costs

- Reduce overhead
  - Send fewer messages, make messages larger
  - Coalesce small messages into large ones

- Reduce delay
  - HW implementor: improve communication architecture
  - Application writer: exploit locality in data distributions

- Reduce contention

- Increase overlap
  - HW implementation: multi-threading, pre-fetching, out-of-order execution
  - Application: asynchronous communication
  - Requires additional concurrency in application (more concurrency than number of processors)

Summary: optimizing communication

- Inherent vs. artifactual communication
  - Artifactual communication depends on the machine
  - Often as important to performance as inherent communication

- Improving program performance:
  - Identify and exploit locality: communicate less
    - increase arithmetic intensity
  - Reduce overhead (few, large messages)
  - Reduce contention
  - Maximize overlap (hide latency so as to not incur cost)