Portable and productive high-performance computing

Eka Palamadai

What is high performance computing?

High-performance computing (HPC) is the use of super computers and parallel processing techniques for solving complex computational problems. HPC

techopedia.com

> *High Performance Computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.*
>
> *InsideHPC.com*

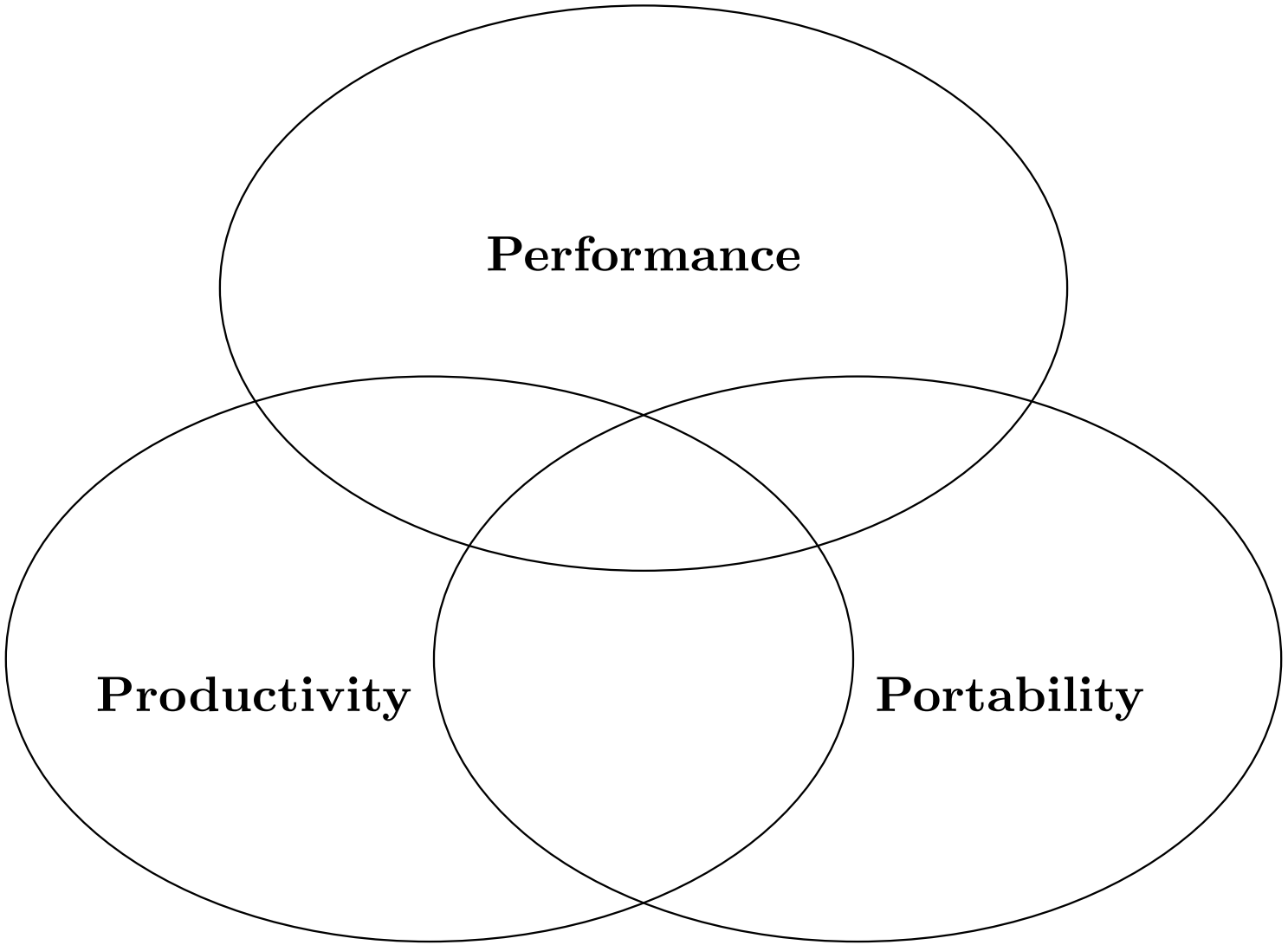
High Performance Computing (HPC) allows scientists and engineers to solve complex science, engineering, and business problems using applications that require high bandwidth, enhanced networking, and very high compute capabilities. AWS allows you to increase the speed of

An expanded definition

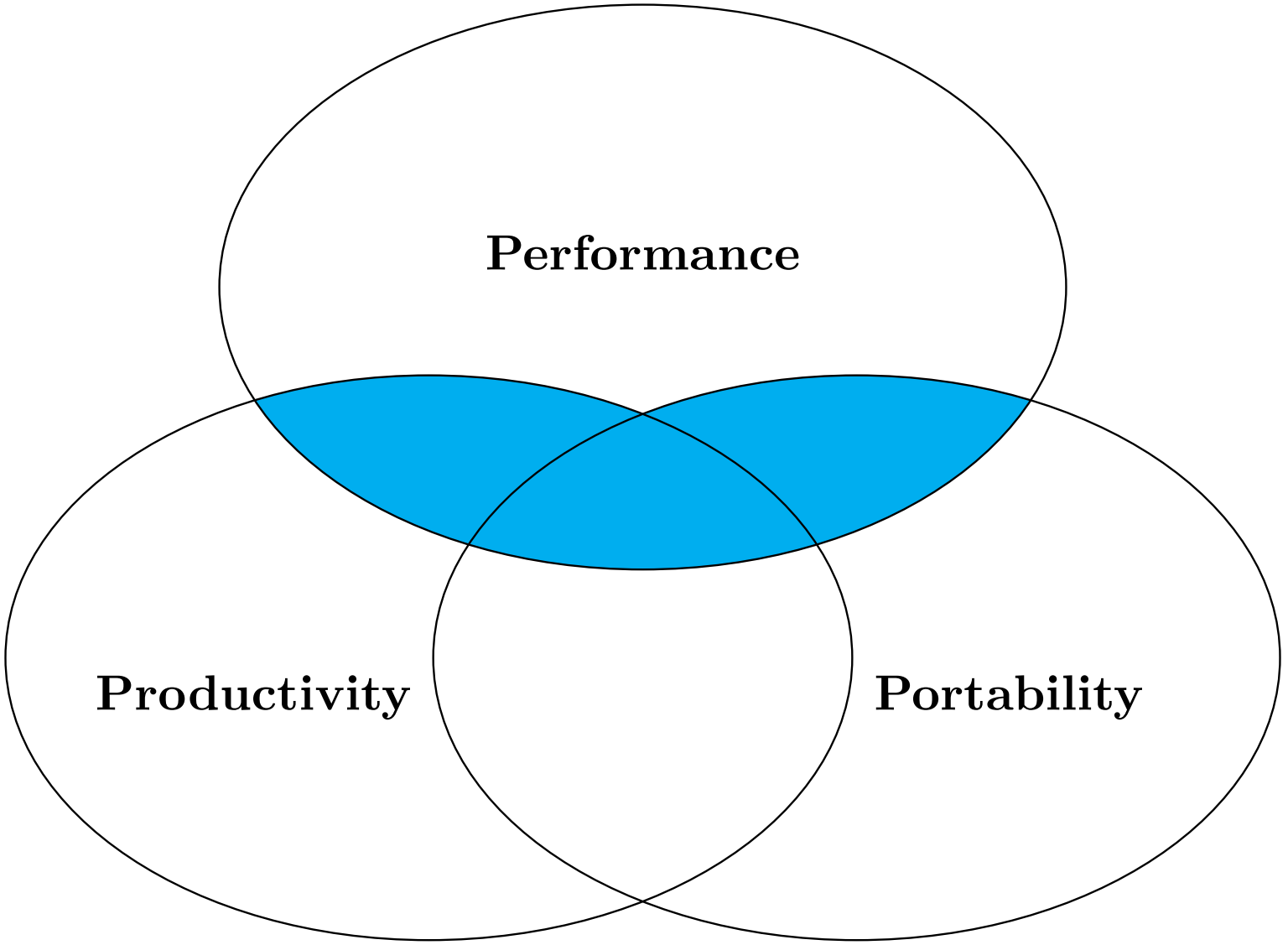
Computing that delivers “high” performance on any given machine configuration

Ideal expectation from HPC

[McCool et al, Structured Parallel Programming]



Focus of this thesis



Performance portability

Write code once, and execute it at optimal speed on any computer.

Programmer productivity

Write parallel code using simple "abstractions" that hide the complex details of parallel programming.

Thesis outline

1. Achieve performance portability for serial "divide-and-conquer" programs
2. Improve programmer productivity in writing parallel code for the ``star'' class of programs

The 1st part of the thesis

1. Achieve performance portability for serial "divide-and-conquer" programs
2. Improve programmer productivity in writing parallel code for the ``star'' class of programs

How to achieve performance portability?

Choose values for key constants in a program to optimize its performance.

Example 1 : Merge sort in C++ STL

```
template<typename _RandomAccessIter>
  void
  __inplace_stable_sort(_RandomAccessIter __first, _RandomAccessIter __last)
  {
    if (__last - __first < 15) {
  __insertion_sort(__first, __last);
  return;
    }
    _RandomAccessIter __middle = __first + (__last - __first) / 2;
    __inplace_stable_sort(__first, __middle);
    __inplace_stable_sort(__middle, __last);
    __merge_without_buffer(__first, __middle, __last,
                __middle - __first,
                __last - __middle);
  }
```

Example 2 : Matrix multiplication

Given 2 matrices A and B to multiply, divide A and B into smaller sizes such that the multiplication optimally uses the memory hierarchy, vectorization, and other resources, and results in faster runtime.

Solution 1 : Handtune

Find values for the key constants on each machine configuration by trial-and-error

Pitfalls

- Error prone
- Time consuming
- Hard to hand-tune 2 or more constants simultaneously
- The original programmer might not be around

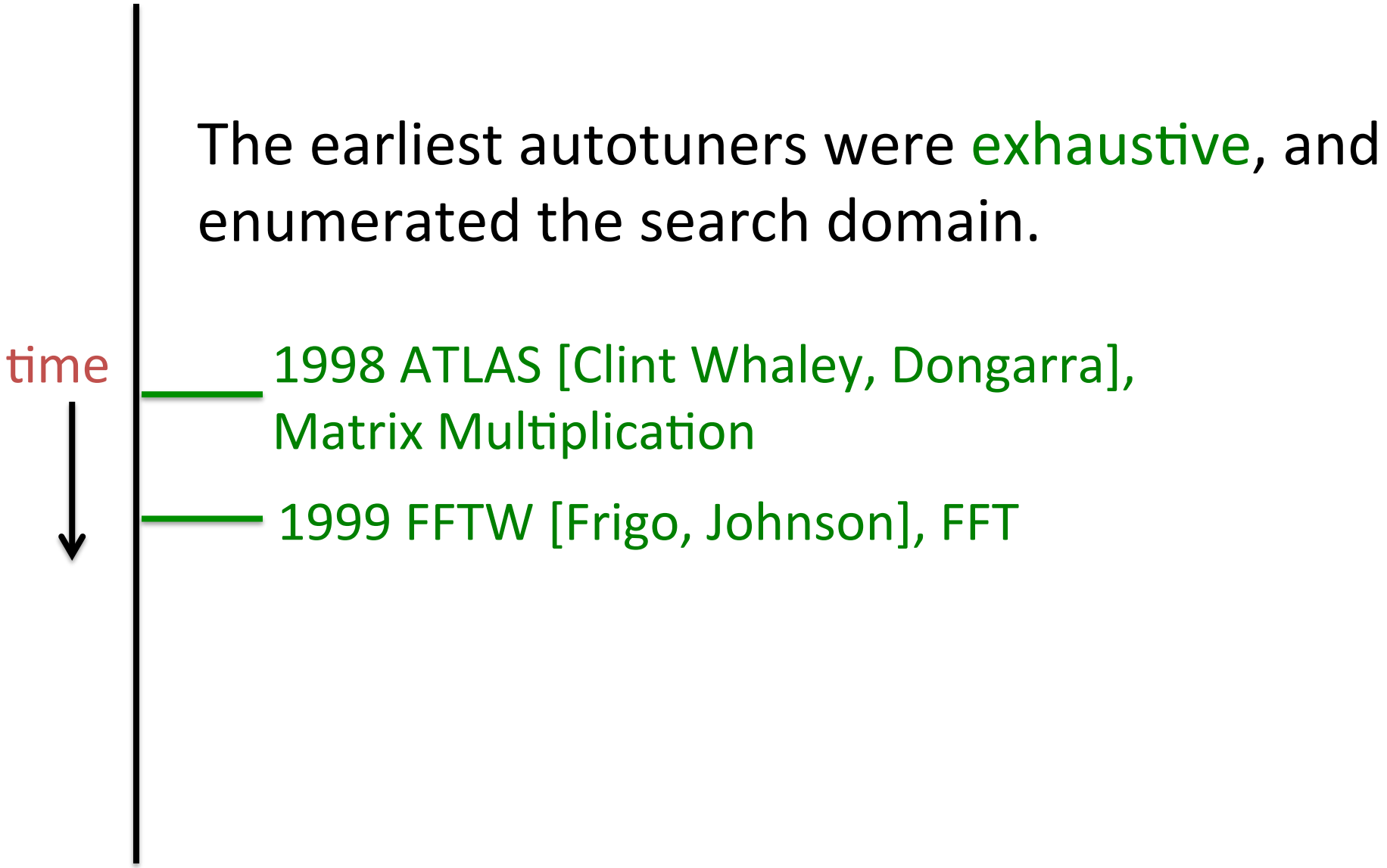
Solution 2 : Autotune

Automatically find values for the key constants for a given machine configuration.

Benefits :

Avoids the pitfalls in handtuning.

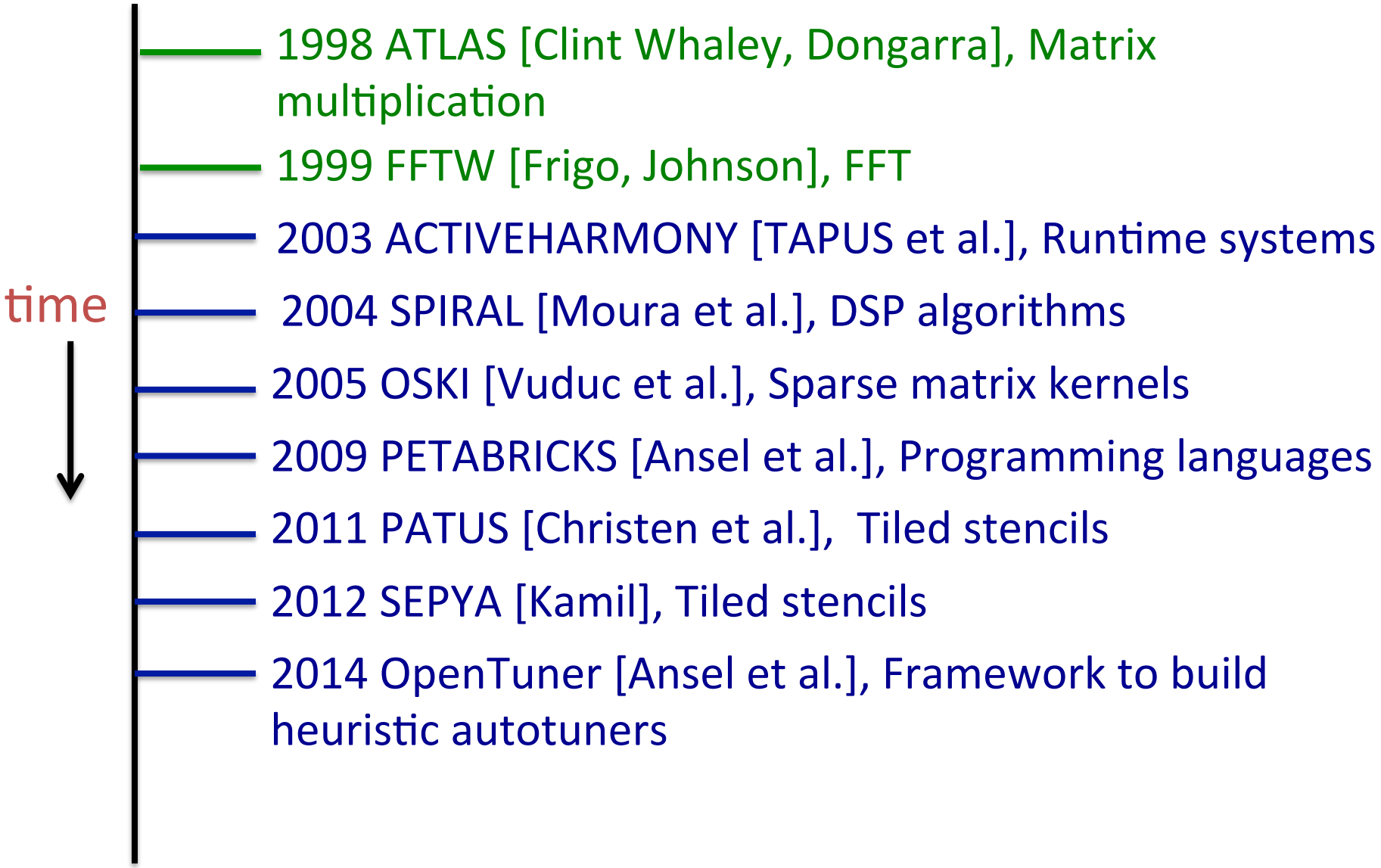
A brief history of autotuners



Heuristic autotuners

The later autotuners were heuristic, and used heuristics and machine-learning based methods to speedup autotuning.

The trend toward heuristic autotuners



Back to exhaustive autotuning

Contributions

Exhaustive autotuning with “pruning” can be more effective than heuristic autotuning for divide-and-conquer “stencil” computations.

Contributions

Ztune : A pruned-exhaustive autotuner for serial, divide-and-conquer codes.

[joint work with Maryam Mehri, Charles Leiserson]

Applications :

- Stencil computations
- Matrix-vector product
- Matrix multiplication

The 2nd part of the thesis

1. Achieve performance portability for serial "divide-and-conquer" programs
2. Improve programmer productivity in writing parallel code for the ``star'' class of programs

The “Star” class consists of programs

1. that execute an associative computation over “ordered” data, and
2. can be parallelized by splitting its computation into a sequence of 3 subcomputations.

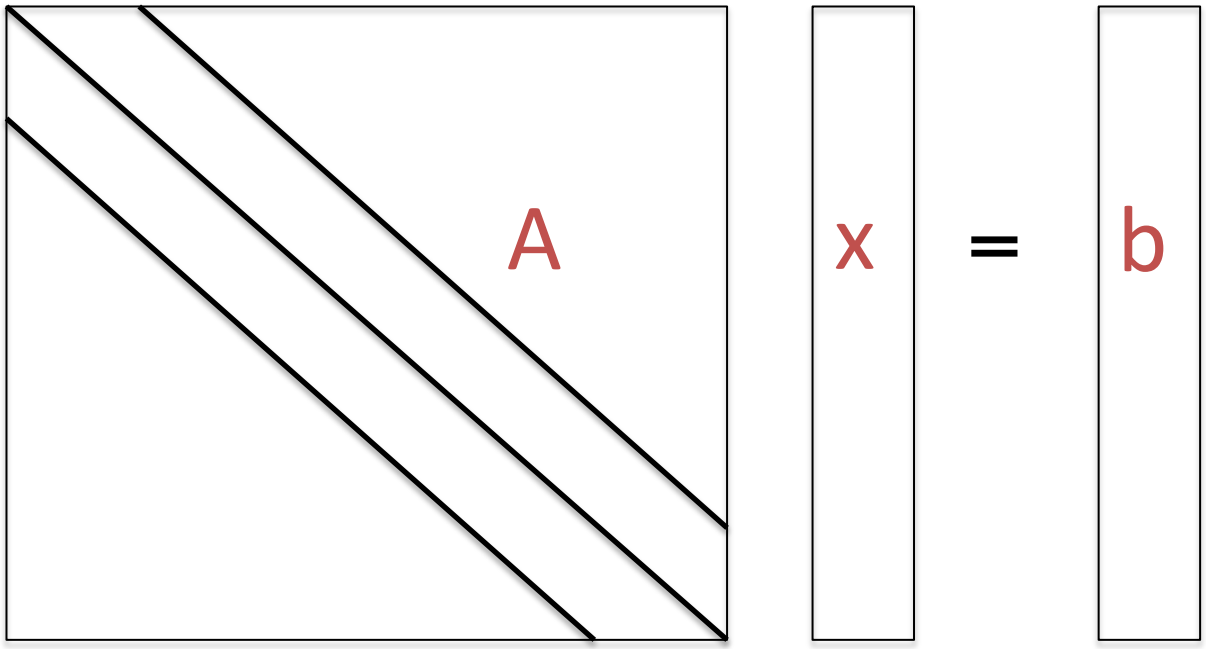
Many seemingly different programs fall in the star class

- Solve Ax = b, where A is a symmetric, diagonally-dominant tridiagonal matrix, and b, x are column vectors.
- Segment graphs and images using "watershed cuts"
- Sample sort
- All-prefix-sums, and its applications [Blelloch, 1990]
  - Radix sort
  - Solving recurrences
  - Dynamic processor allocation, and many more

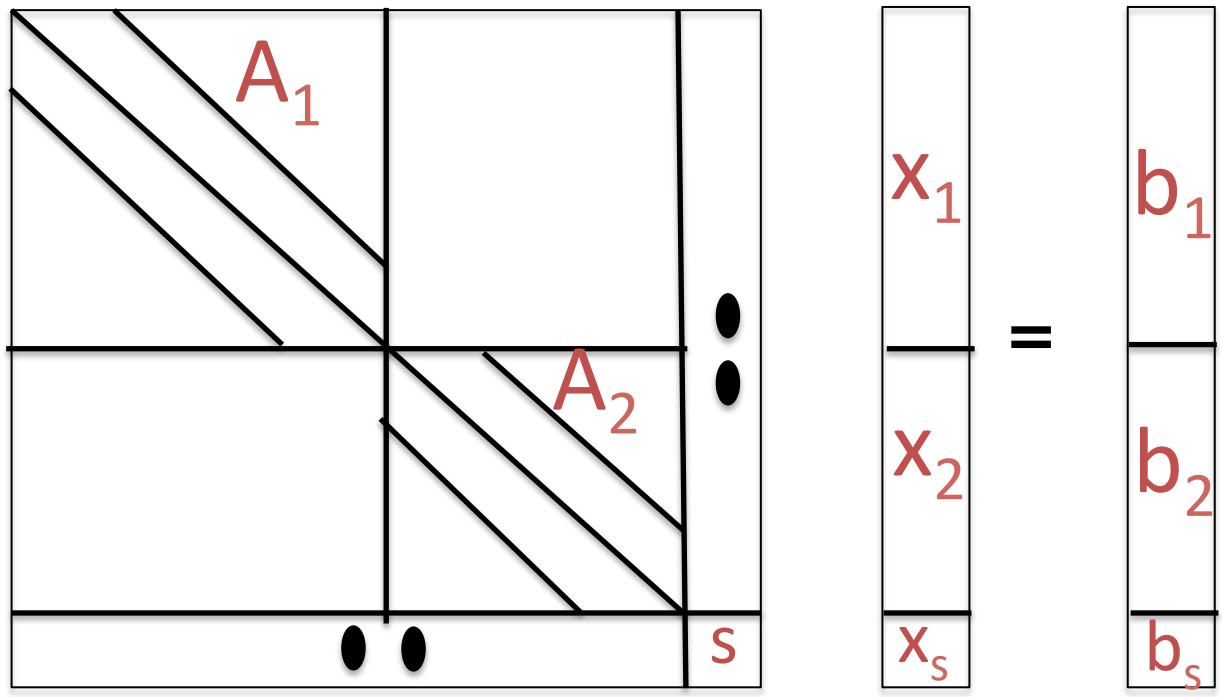
Example 1 : Tridiagonal systems

Given p processors, solve $Ax = b$, where A is a symmetric diagonally-dominant tridiagonal matrix, and b, x are column vectors.

A parallel algorithm for 2 processors



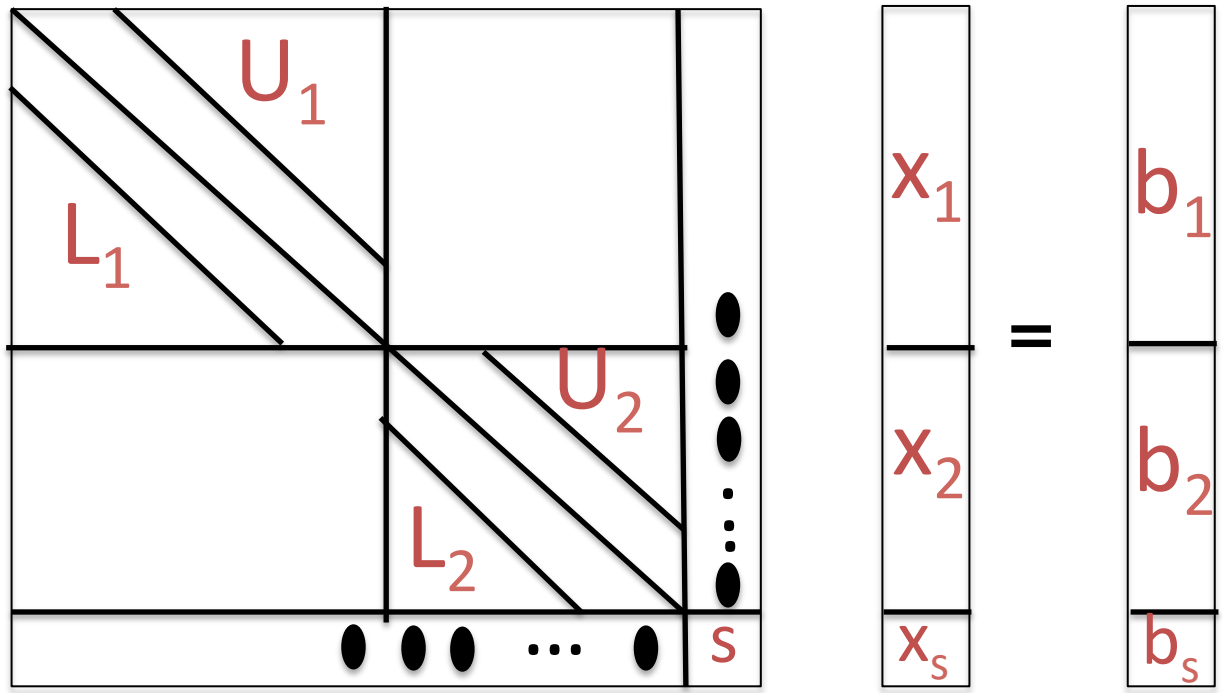
Partition A, x, and b



Trisect A into tridiagonal matrices A_1 , A_2, and a separator s ;

Trisect x into x_1, x_2, and x_s ; Trisect b into b_1, b_2, and b_s

Step 1 : Factorize and forward solve in parallel

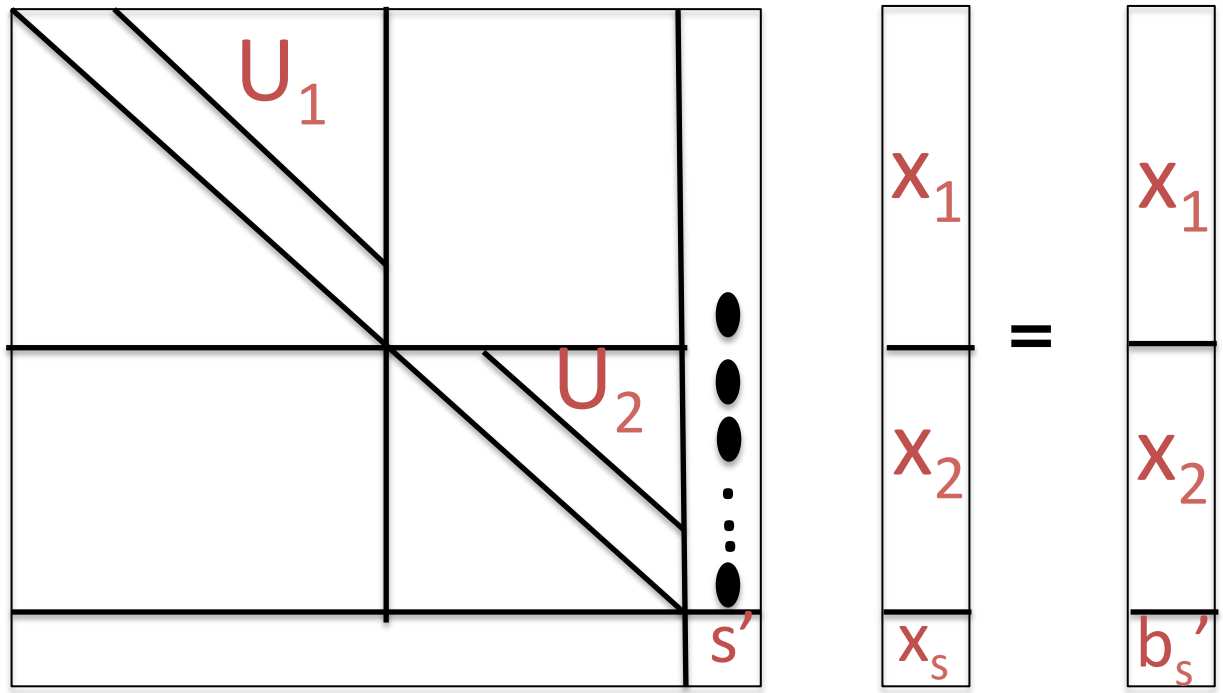


function lsolve ()

Parallel : { Factorize $A_1 = L_1U_1$ and forward solve $L_1x_1 = b_1$;

Factorize $A_2 = L_2U_2$ and forward solve $L_2x_2 = b_2$;}

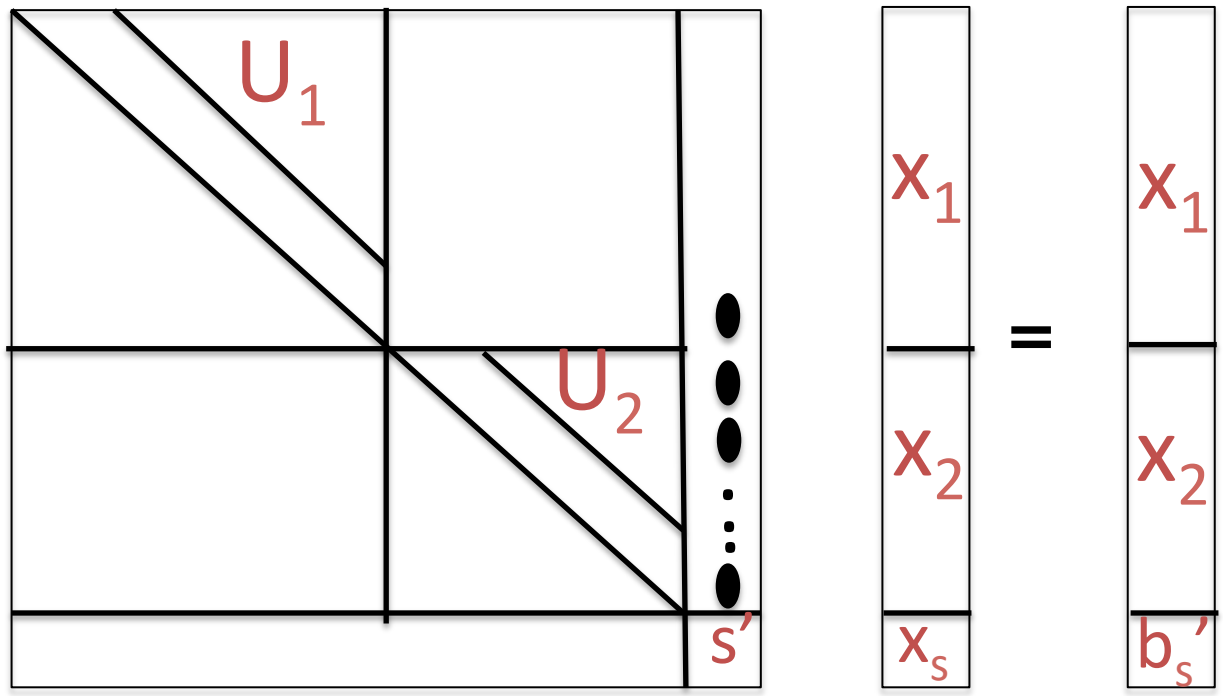
Step 2 : Solve a reduced system in serial



function rsolve ()

Solve the reduced system $s'x_s = b_s'$

Step 3 : Back solve in parallel



function usolve ()

Parallel : { Back solve $U_1x_1 = x_1$; Back solve $U_2x_2 = x_2$;}

Example 2 : Image segmentation using "watershed" cuts

Find segments of "similar" pixels in an image.

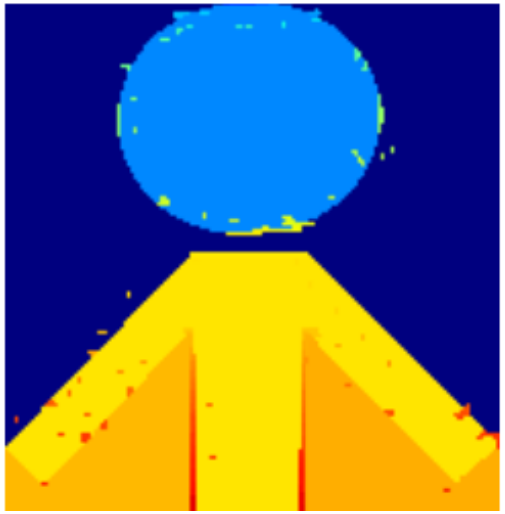
An example input and output images

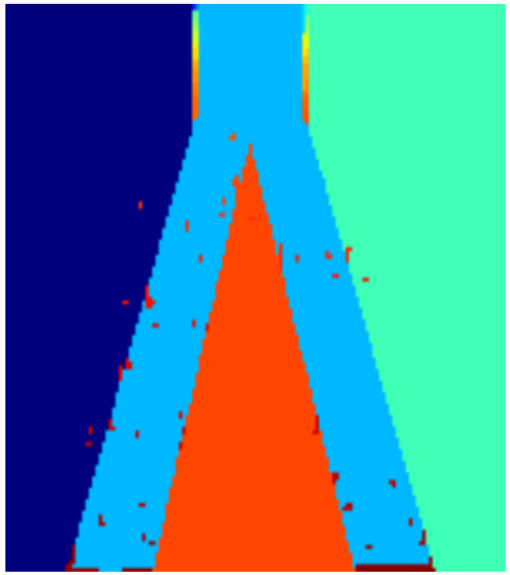
Input

Output

Partition image into 2 subimages

Step 1: Segment the subimages in parallel

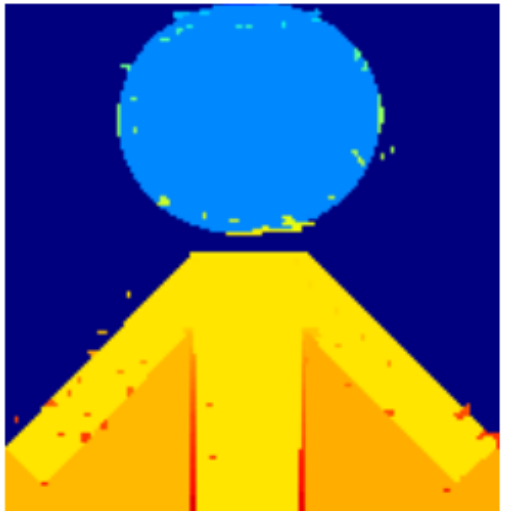




Step 2: Merge the boundaries in serial

Reconcile the segments at the boundaries of the images

Step 3: Update the subimages in parallel



The 3 steps in the execution of the subcomputations

Arrange the ordered data in partitions.

1 Execute the 1st subcomputation in parallel on the partitions.

2a Assemble the results from the 1st step in the order of the partitions

2b Execute the 2nd subcomputation in serial on the assembled data.

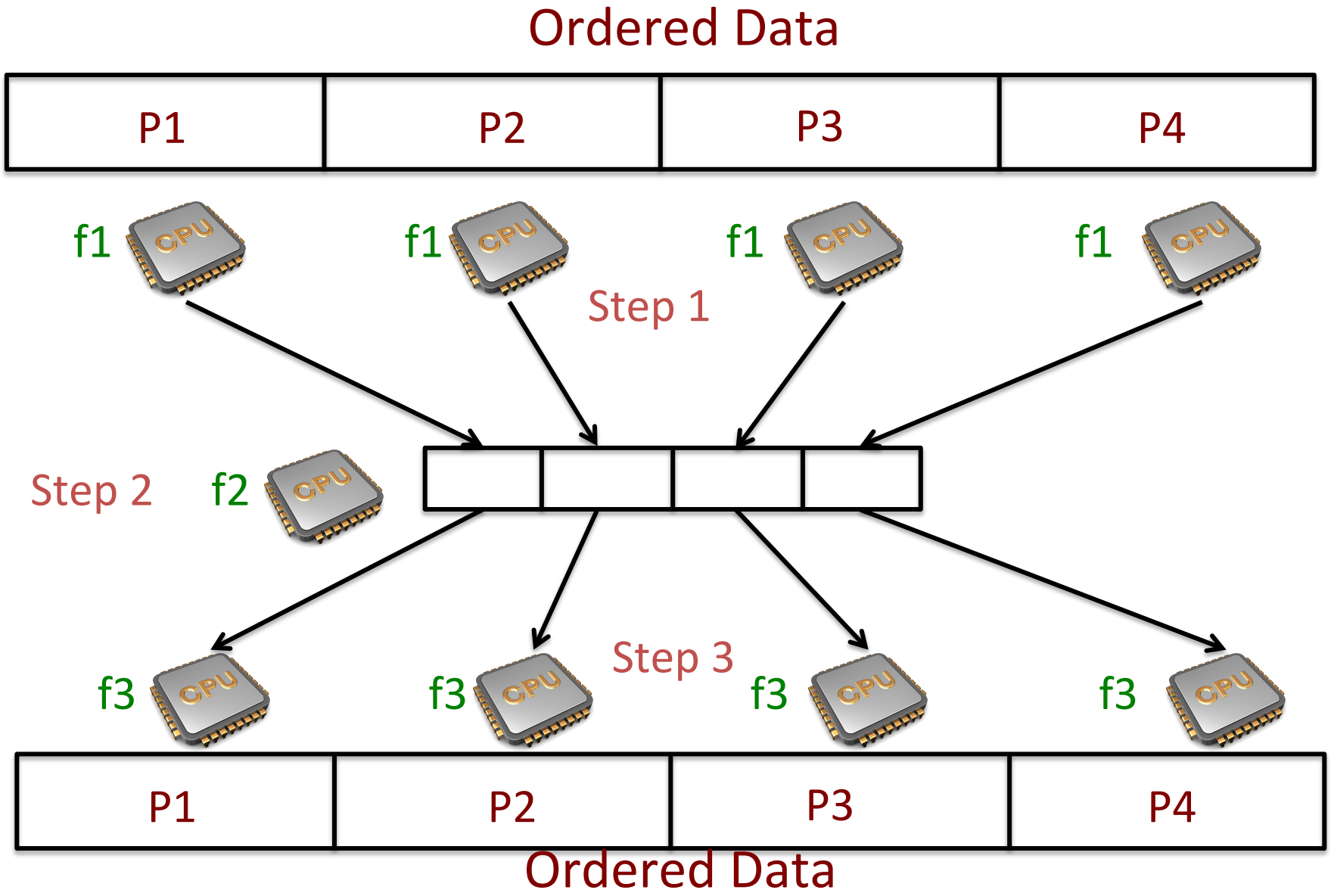
2c Send the results back to the partitions

3 Execute the 3rd subcomputation in parallel on the partitions using the results from the 2nd.

Properties of the 3 subcomputations

- The 3 subcomputations are associative.
- The 2nd subcomputation is noncommutative.
- Communication between two subcomputations is asymptotically smaller than the subcomputation time.

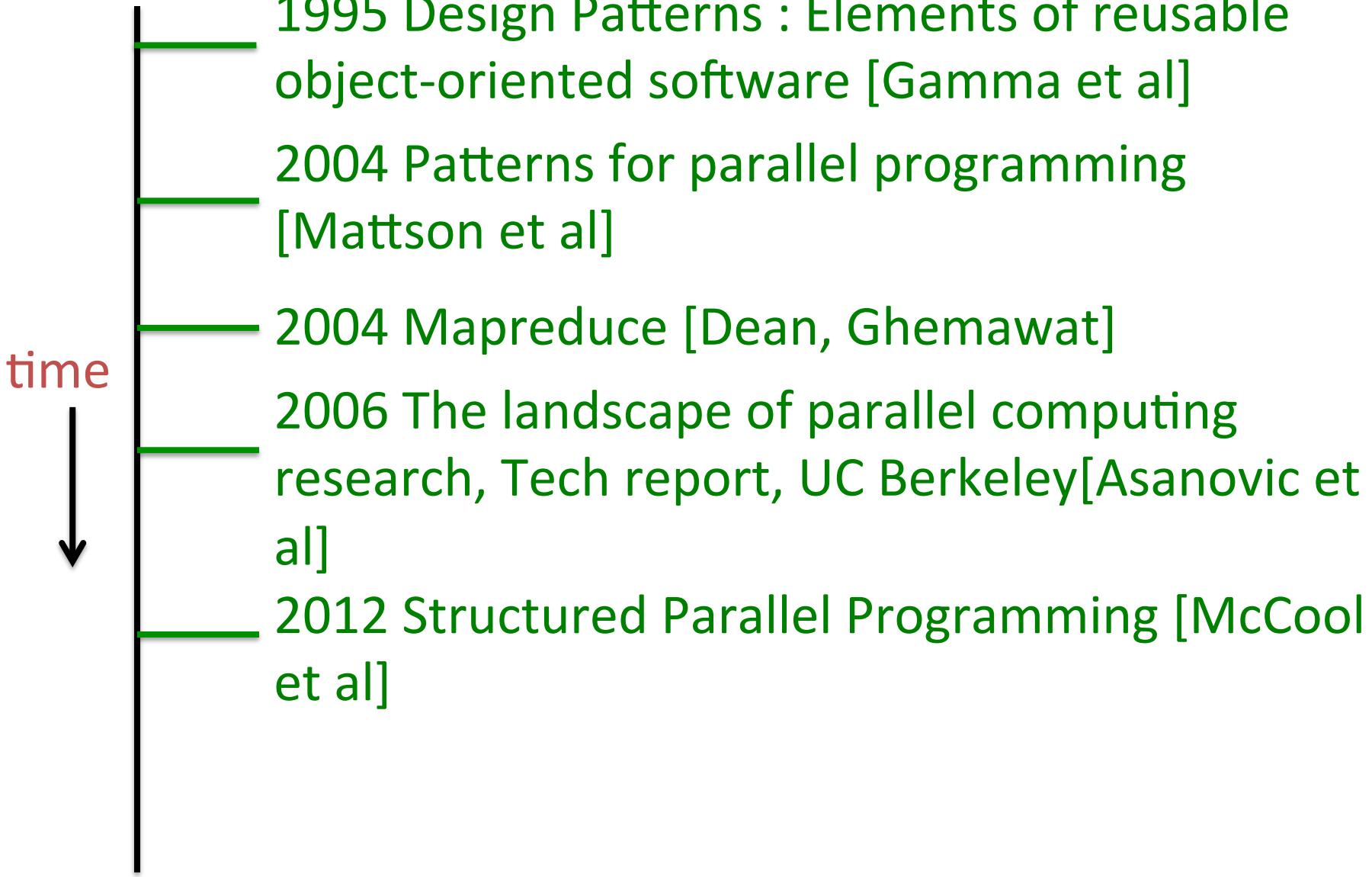
The structure of the star execution



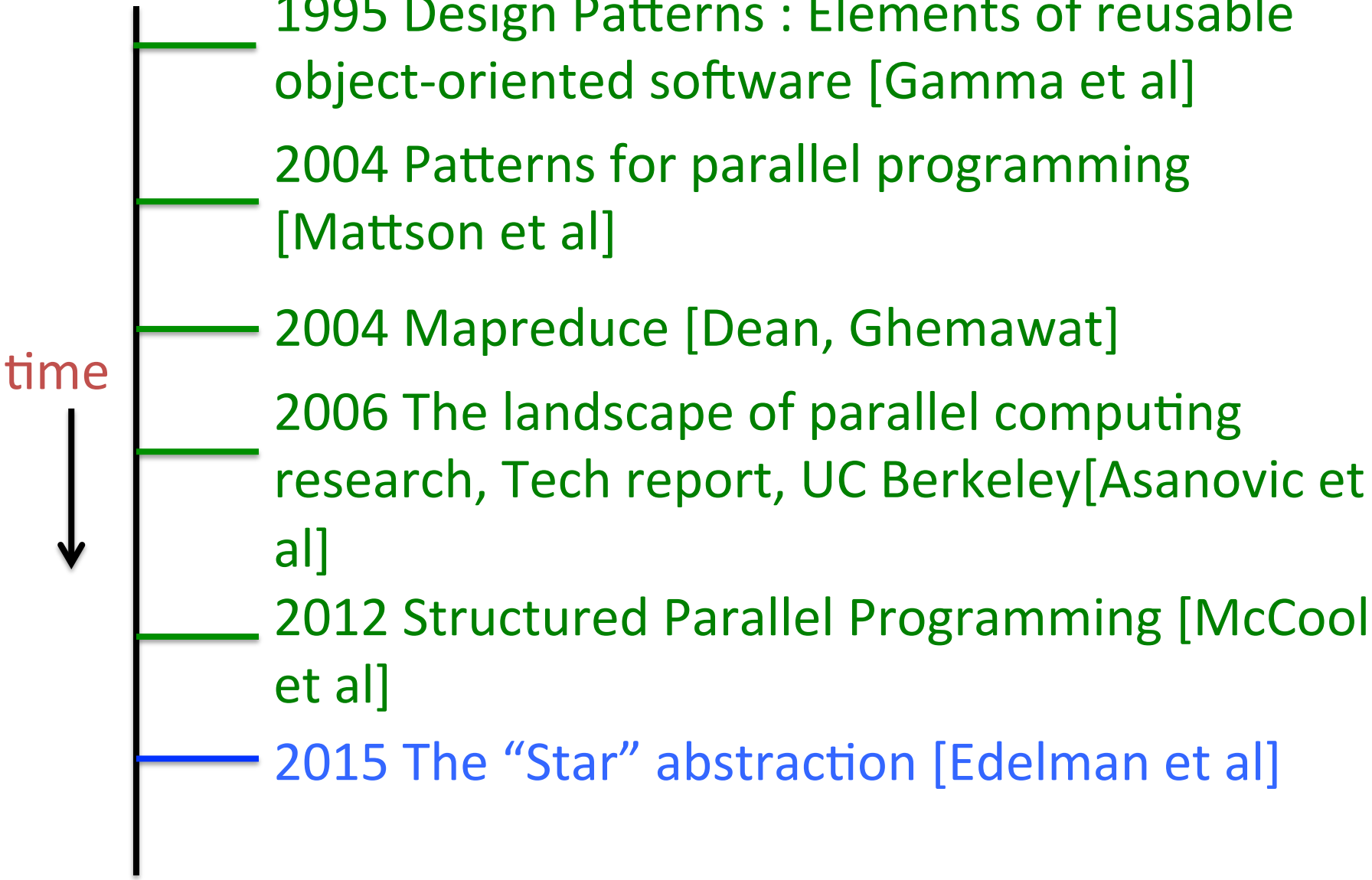
The star class is worth an "abstraction"

- Parallel programs in the star class have the same pattern of computation and communication, which can be abstracted away.
- Programmers can code to a simple abstraction that hides the complexities and low level details of the parallel programming model.

A brief history of abstractions



Contribution



Contributions

- Star : An abstraction for the star class of parallel programs. [joint work with Andreas Noack, Alan Edelman]
- Identifying the star structure in seemingly different programs.
- Parallel algorithms for solving tridiagonal systems and “watershed cuts” that offer improvements over prior art.

Outline

- Ztune
  - Autotune the stencil code in Pochoir stencil compiler.[SPAA 2011]
- Star
  - A “Wasp” algorithm for parallel watershed cuts
- Future work

Outline

- Ztune
  - Autotune the stencil code in Pochoir stencil compiler.[SPAA 2011]
- Star
  - A “Wasp” algorithm for parallel watershed cuts
- Future work

What is a stencil?

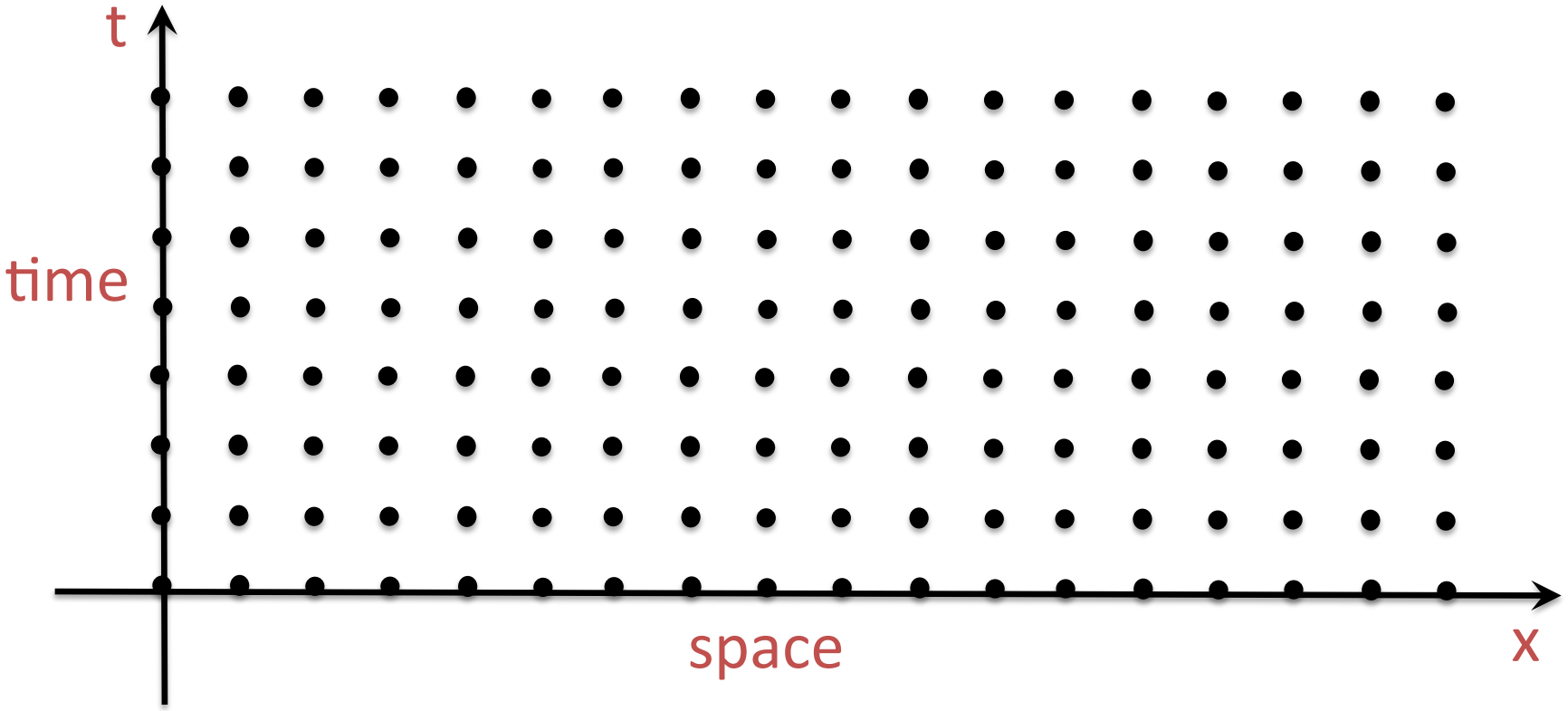
A mathematical equation that defines the value of a grid point in a spatial grid at time t, as a function of the values of its neighbors at recent times before t.

A stencil example

1D heat equation

$$\frac{\delta u}{\delta t} = \alpha \frac{\delta^2 u}{\delta x^2}$$

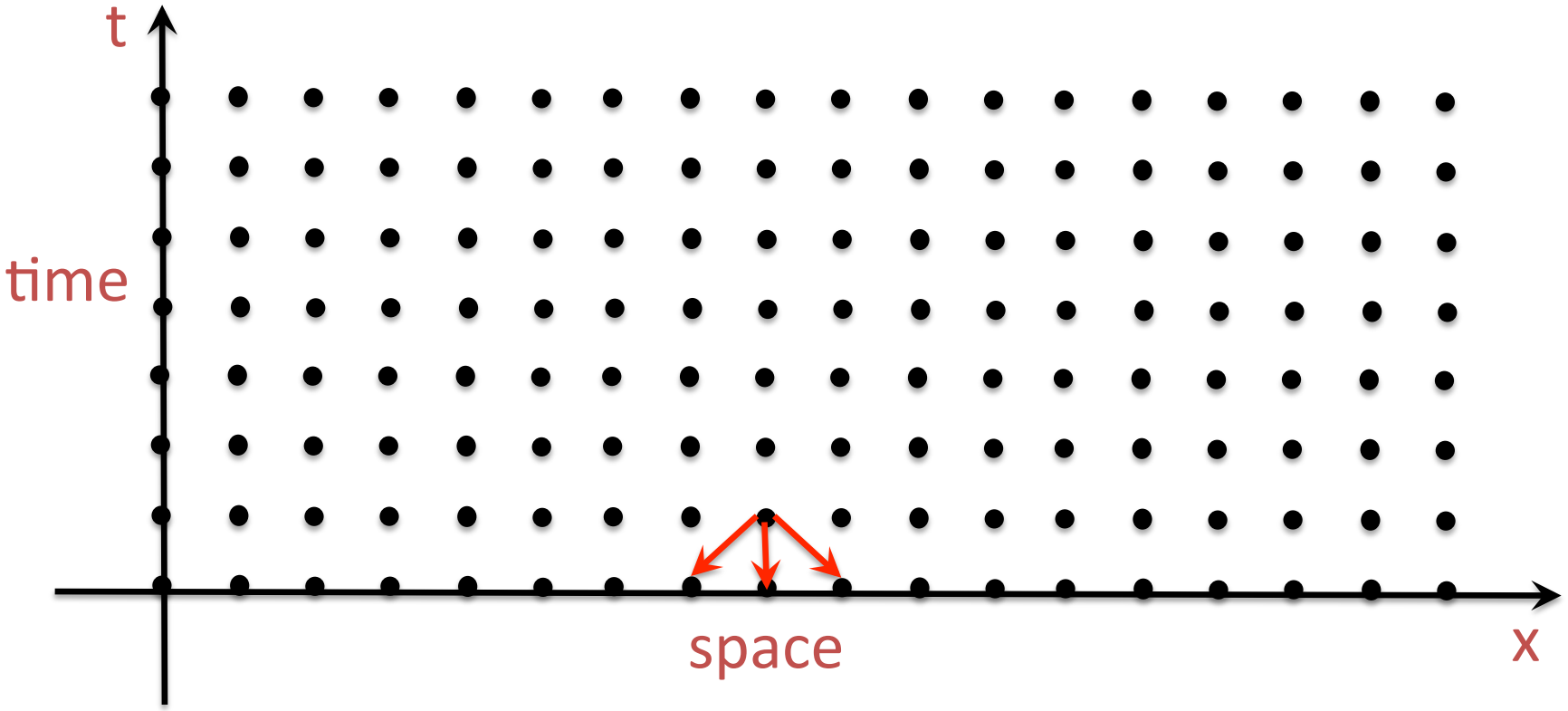
Let u(t, x) be temperature at (t, x), and α be the thermal coefficient.



The 1D heat stencil

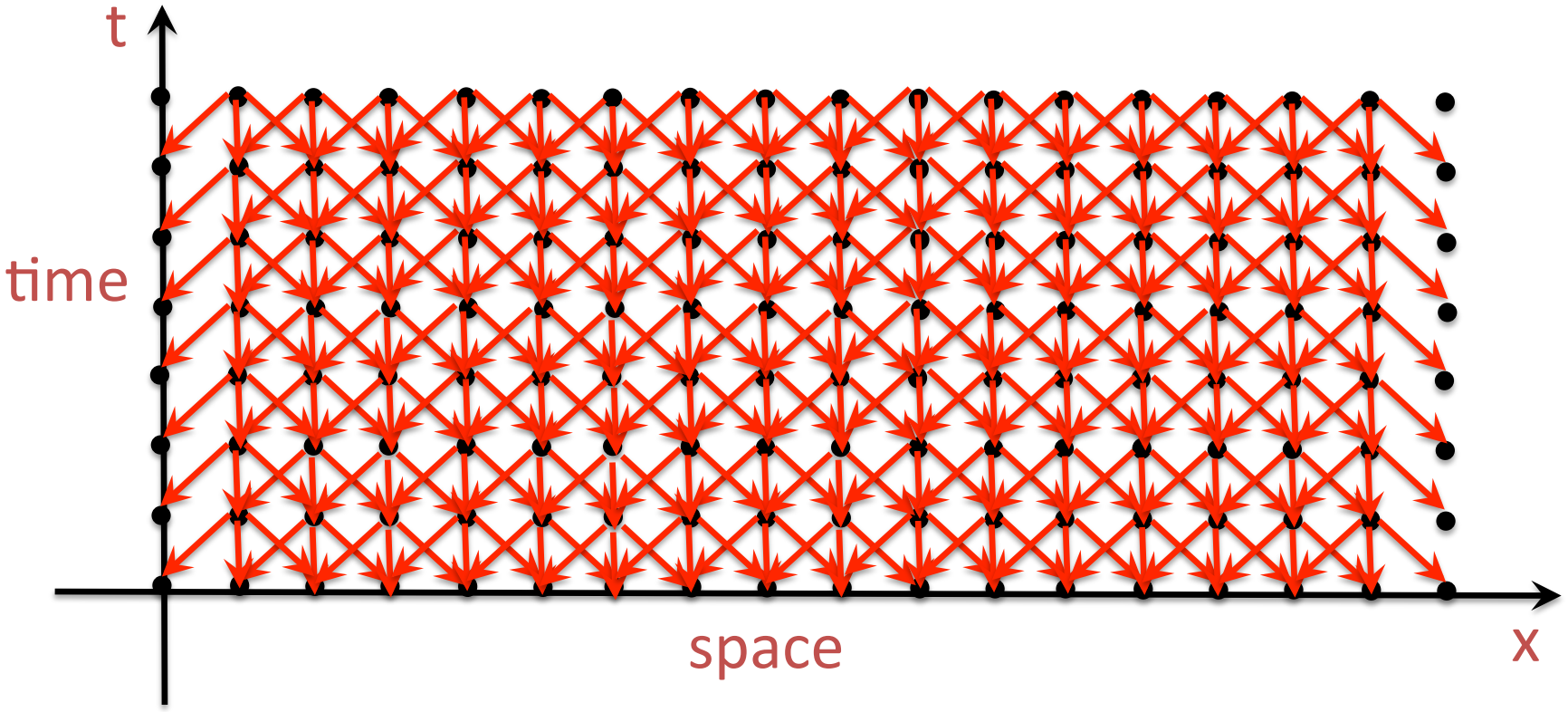
Stencil is the discretized heat equation

$$u(t, x) = f(u(t-1, x+1), u(t-1, x), u(t-1, x-1), \alpha)$$



What is stencil computation?

Compute the stencil at all grid points in the space-time grid



Evidence for the claim

Exhaustive autotuning with “pruning” can be more effective than heuristic autotuning for divide-and-conquer “stencil” computations.

Comparison with heuristic autotuning

OpenTuner [Ansel et al., PACT 2014]

- a state-of-the-art heuristic autotuner
- can be used to autotune divide-and-conquer stencil codes

Ten stencil benchmarks in Pochoir

Name	Dims	Grid size	Time steps
APOP	1	2,000,000	524288
Heat1	2	1000 X 2000	512
Heat2	2p	100 X 20,000	8192
Life	2p	2000 X 3000	1024
Heat3	2p	4000^2	1024
Heat4	2p	4096^2	512
Heat5	2p	$10{,}000^2$	4096
LBM	3	100 X 100 X 130	64
Wave	3	1000^3	64
Heat6	4	70^4	32

Machine specifications

	Nehalem	Ivy Bridge
CPU	Xeon X5650	Xeon E5-2695 v2
Clock	2.66 GHz	2.40 GHz
Hyperthreading	Disabled	Enabled
L1 data cache	32KB	32KB
L2 cache	256KB	256KB
L3 cache	12MB	30MB
Compiler	ICC 13.1.1	ICC 13.1.1
Linux kernel	3.13.0	3.13.0
Advanced Vector Extensions	No	Yes

Performance metric 1 : Quality

$$\text{Quality} : \frac{\text{Runtime of autotuned code}}{\text{Runtime of default code}}$$

Quality of OpenTuner and Ztune

Name	Nehalem		Ivy Bridge	
	OpenTuner	Ztune	OpenTuner	Ztune
APOP	0.98	0.98	1.32	0.96
Heat1	0.93	0.94	0.92	0.92
Heat2	0.84	0.88	0.84	0.89
Life	0.97	0.97	0.99	0.99
Heat3	0.94	0.96	0.89	0.91
Heat4	1.00	1.00	0.67	0.60
Heat5	1.16	0.92	1.00	0.87
LBM	0.84	0.98	0.93	0.97
Wave	3.57	0.96	1.45	0.88
Heat6	0.98	0.90	0.99	0.87

Autotuning times of OpenTuner and Ztune

Name	Nehalem		Ivy Bridge	
	OpenTuner	Ztune	OpenTuner	Ztune
APOP	16 h	2m	16 h	35s
Heat1	16 h	32s	16 h	20s
Heat2	16 h	5m	16 h	7m
Life	16 h	7m	16 h	4m
Heat3	16 h	3m	16 h	1m
Heat4	16 h	2m	16 h	47s
Heat5	16 h	12m	16 h	7m
LBM	16 h	38s	16 h	34s
Wave	16 h	17m	16 h	52m
Heat6	16 h	4m	16 h	6m

Autotuning times to achieve the same quality on Ivy Bridge

Name	Ztune	OpenTuner
APOP	0.58 m	37.93 h
Heat1	0.33 m	0.15 h
Heat2	6.58 m	4.17 h
Life	4.40 m	1.32 h
Heat3	1.09 m	2.66 h
Heat4	0.79 m	> 100 h
Heat5	6.55 m	48.75 h
LBM	0.56 m	0.65 h
Wave	51.90 m	> 100 h
Heat6	5.75 m	> 100 h

Performance metric 2 : Tuning speed

$$\text{Tuning speed} : \frac{\text{Tuning time of autotuner}}{\text{Runtime of tuned code}}$$

Tuning speed of Ztune

Name	Nehalem	Ivy Bridge
APOP	0.05	0.03
Heat1	14.49	11.50
Heat2	4.86	7.77
Life	18.50	15.48
Heat3	5.48	2.65
Heat4	2.03	2.41
Heat5	0.93	0.70
LBM	2.95	2.54
Wave	1.18	3.82
Heat6	11.25	18.17

Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP

Pochoir's TRAP Algorithm

[SPAA 2011]

- A cache-efficient divide-and-conquer stencil algorithm.
- Modeled on the trapezoidal decomposition framework. [Frigo and Strumpen, ICS 2005]

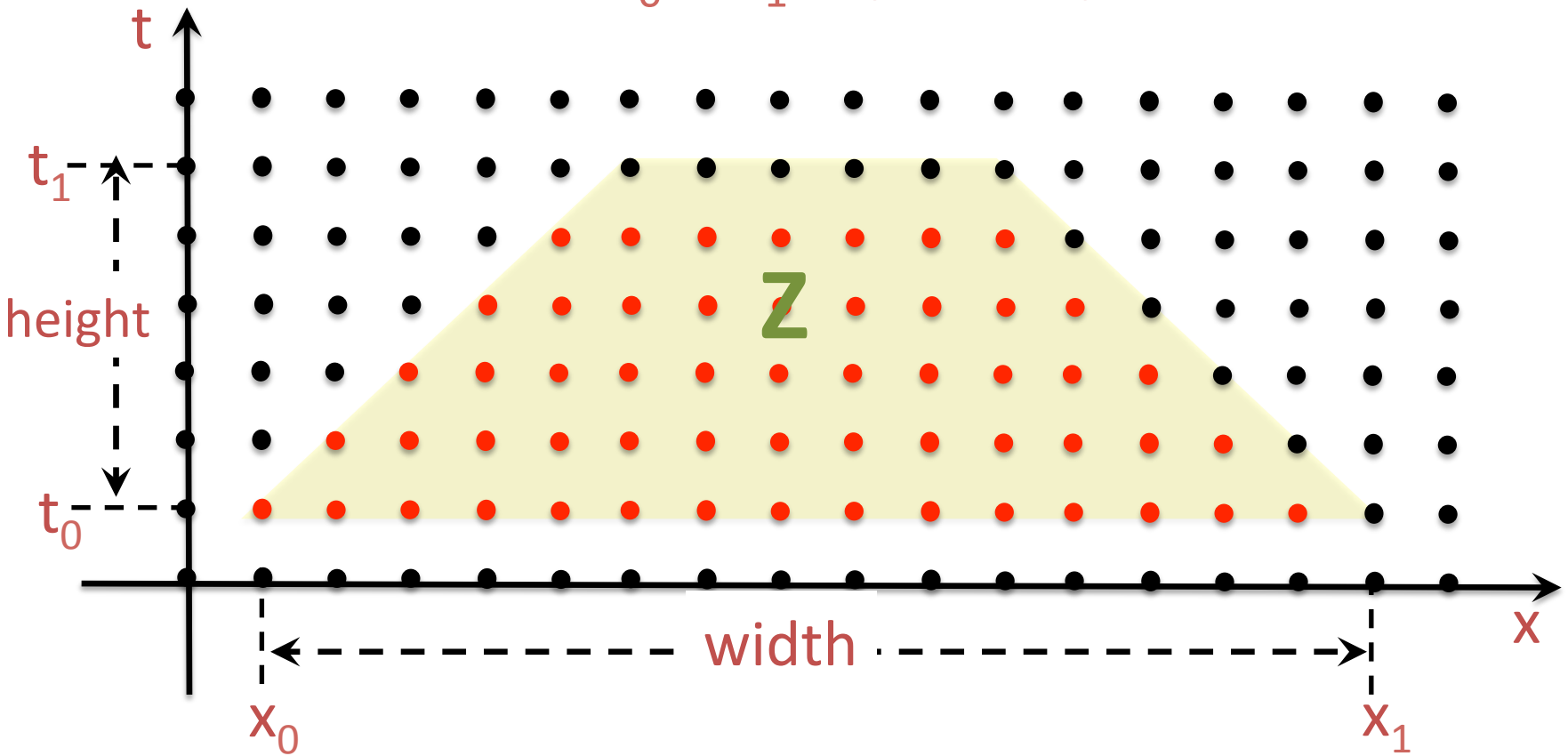
TRAP illustrated

TRAP recursively traverses trapezoidal regions of space-time points (t, x) such that

$$t_0 \le t < t_1$$

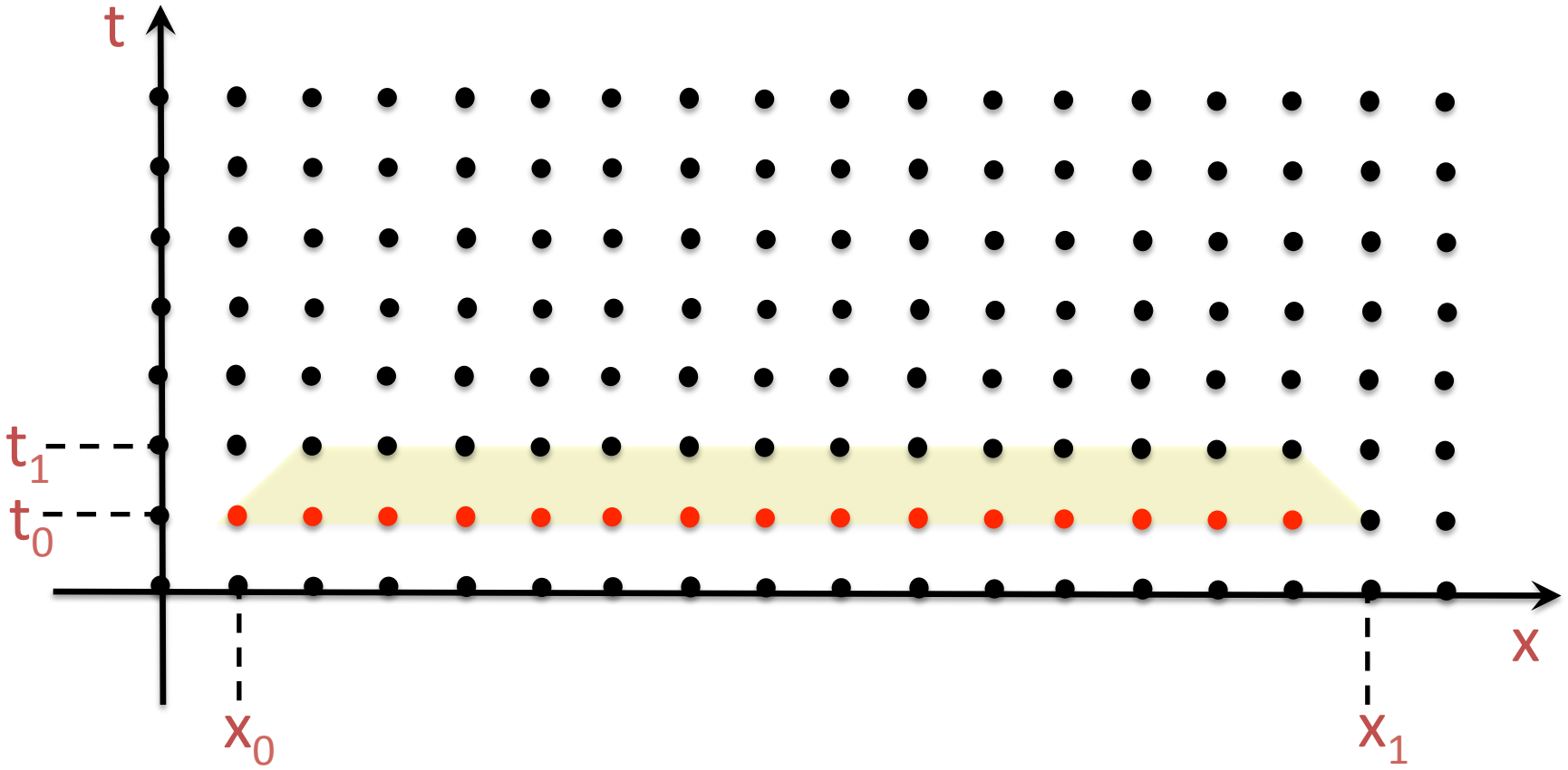
$$x_0 + dx_0(t - t_0) \le x < x_1 + dx_1 (t - t_0)$$

$$dx_0, dx_1 \in \{1, 0, -1\}$$



Base case

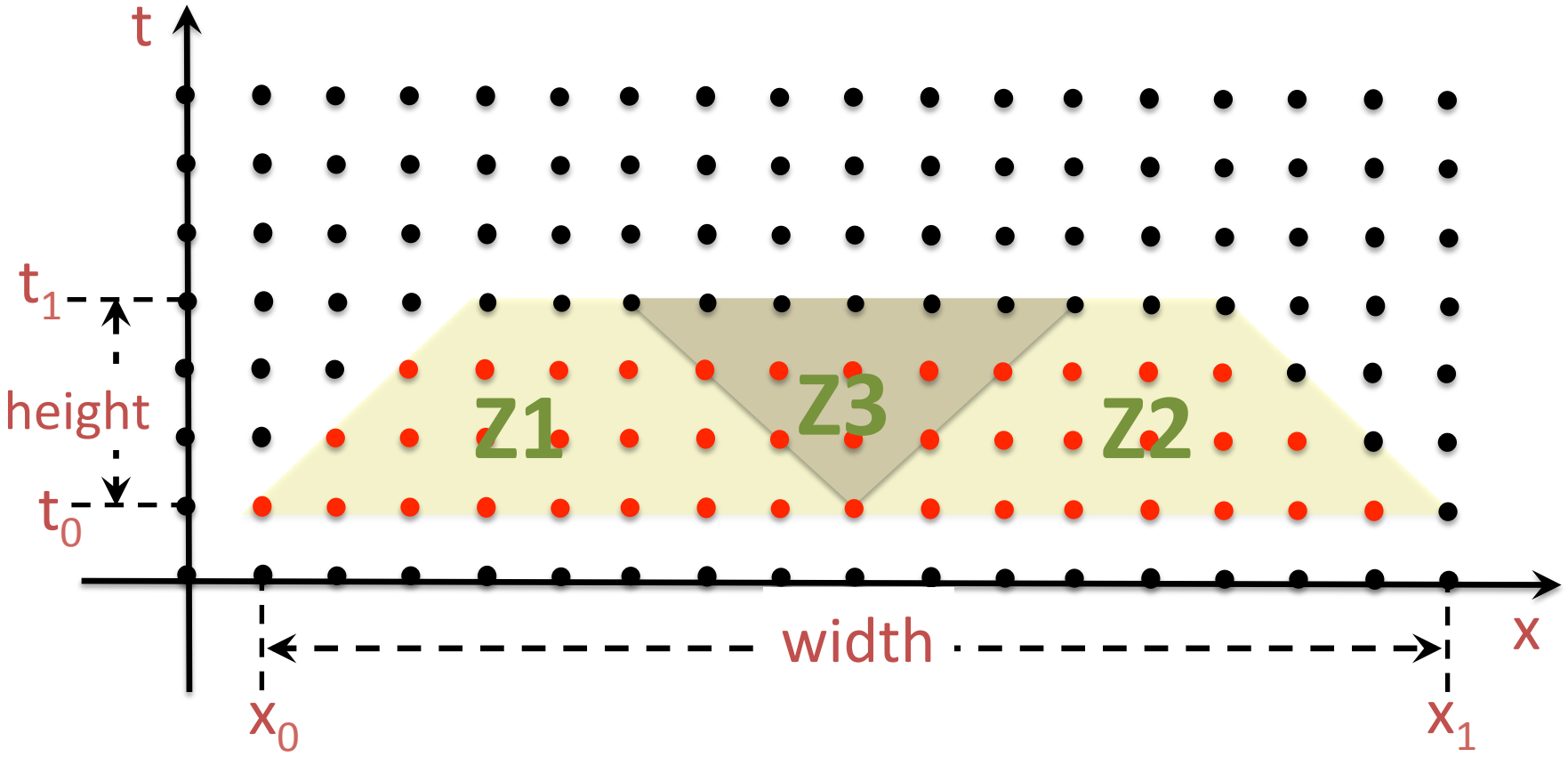
If height = 1, compute stencil at all grid points in the trapezoid.



Space cut

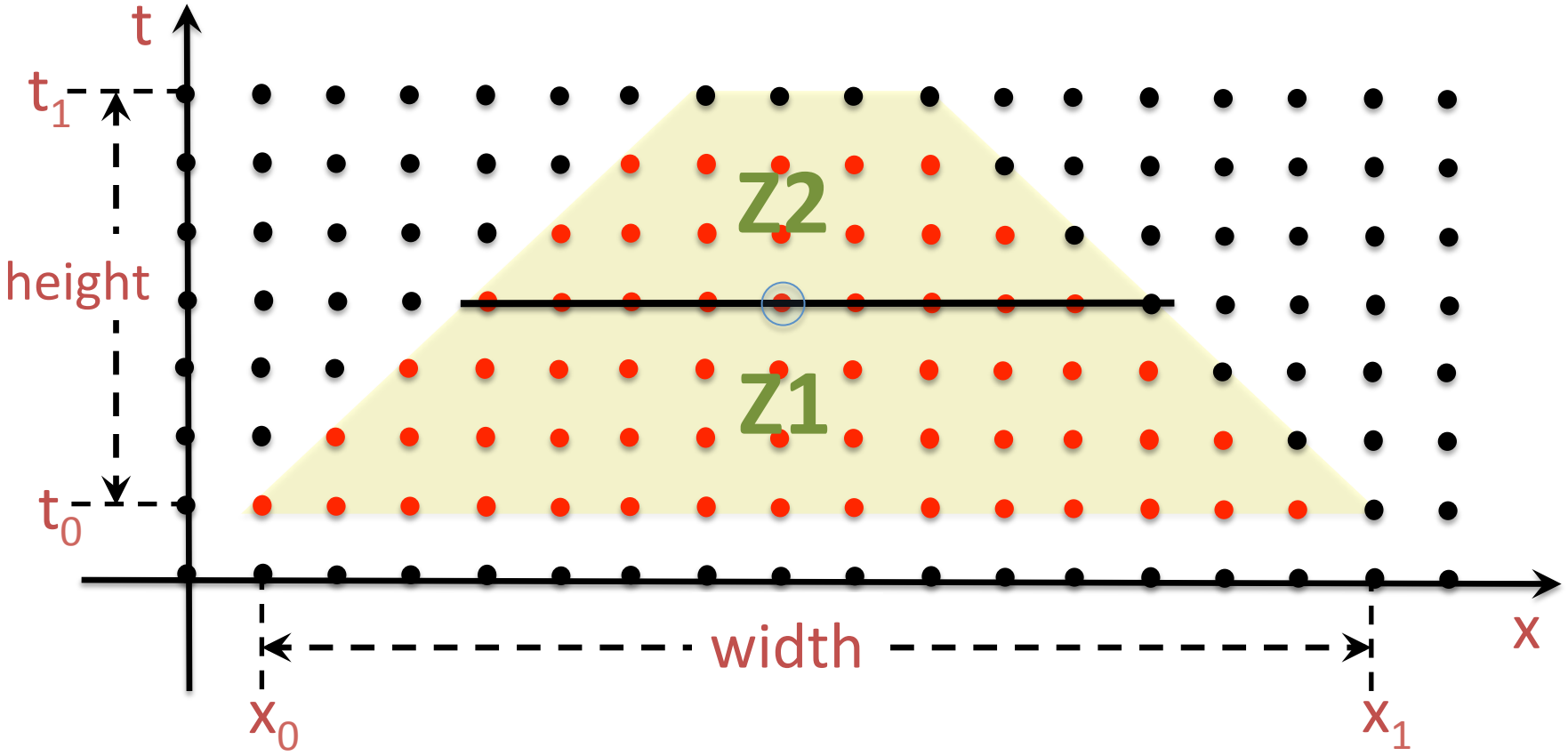
If width ≥ 4 height, cut the trapezoid into 3 pieces.

Traverse Z1 and Z2 first, and then Z3.



Time cut

If width < 4 height, cut the trapezoid with a horizontal line through the center. Traverse Z1 first, and then Z2.



Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - The search domain of base-case sizes

The 2 “kernels” of TRAP

- Kernel is the code that executes the base-case stencil computation on a given space-time grid
- TRAP uses 2 kernels
  - A slower kernel at the boundary that checks if a memory access falls off the grid
  - A faster kernel in the interior that does not perform the check
- Cache sizes matter in the interior, but not at the boundary

Autotuning strategy 1 : Search for optimal base-case sizes

- Choose base-case sizes at the interior, and the boundary such that the stencil computation
  - optimally uses the memory hierarchy, vectorization, and other system resources,
  - minimizes function-call overhead, and
  - minimizes the usage of the slower boundary kernel

The search domain of base-case sizes : "Base domain"

- The base domain consists of the set of all possible values for the base-case sizes at the interior and the boundary.
- The size of the base domain is polynomial in the size of the grid.
  - For example, w^2h^2 for a 2 dimensional space-time grid with width w and height h
- A heuristic autotuner can be used to search over the base domain.

Problems with the base domain

- The divide-and-conquer TRAP stencil program creates base-cases of different sizes and shapes.
- Hence the chosen base-case sizes might not optimize the execution of all base-cases.

Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - The search domain of base-case sizes
  - The search domain of divide-and-conquer trees

Division choices

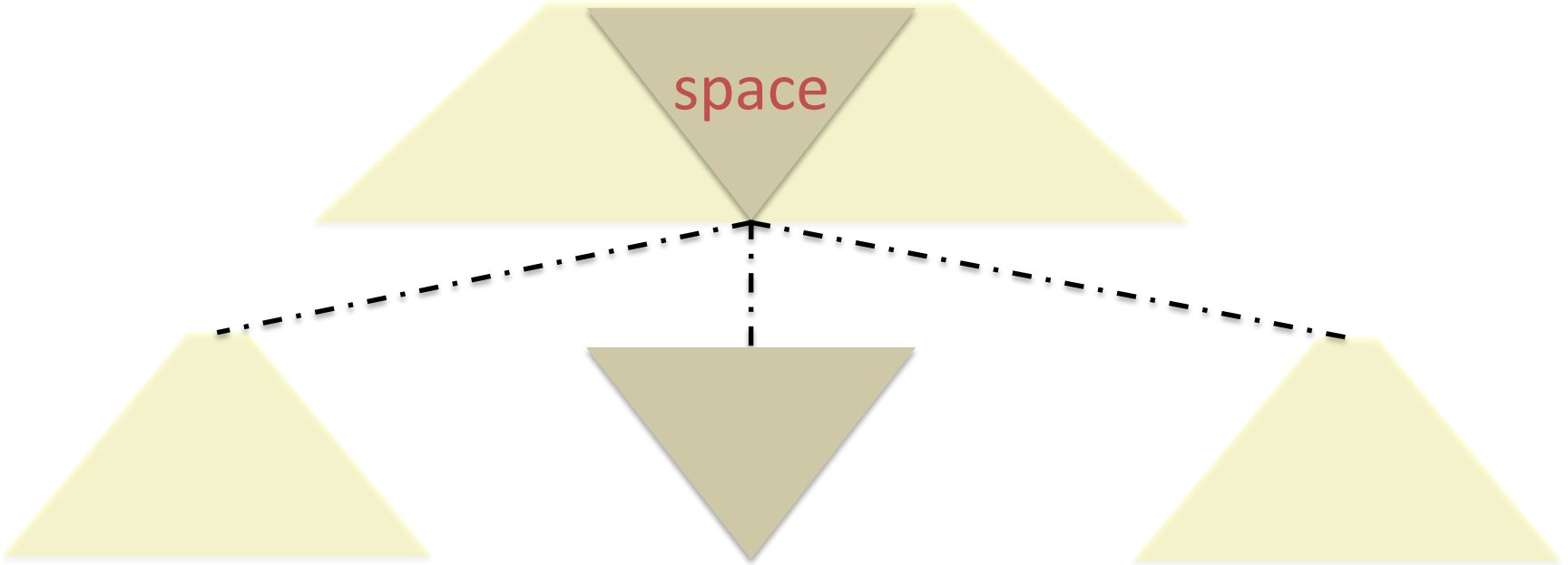
A 2D space-time grid has at most 3 division choices, namely

time

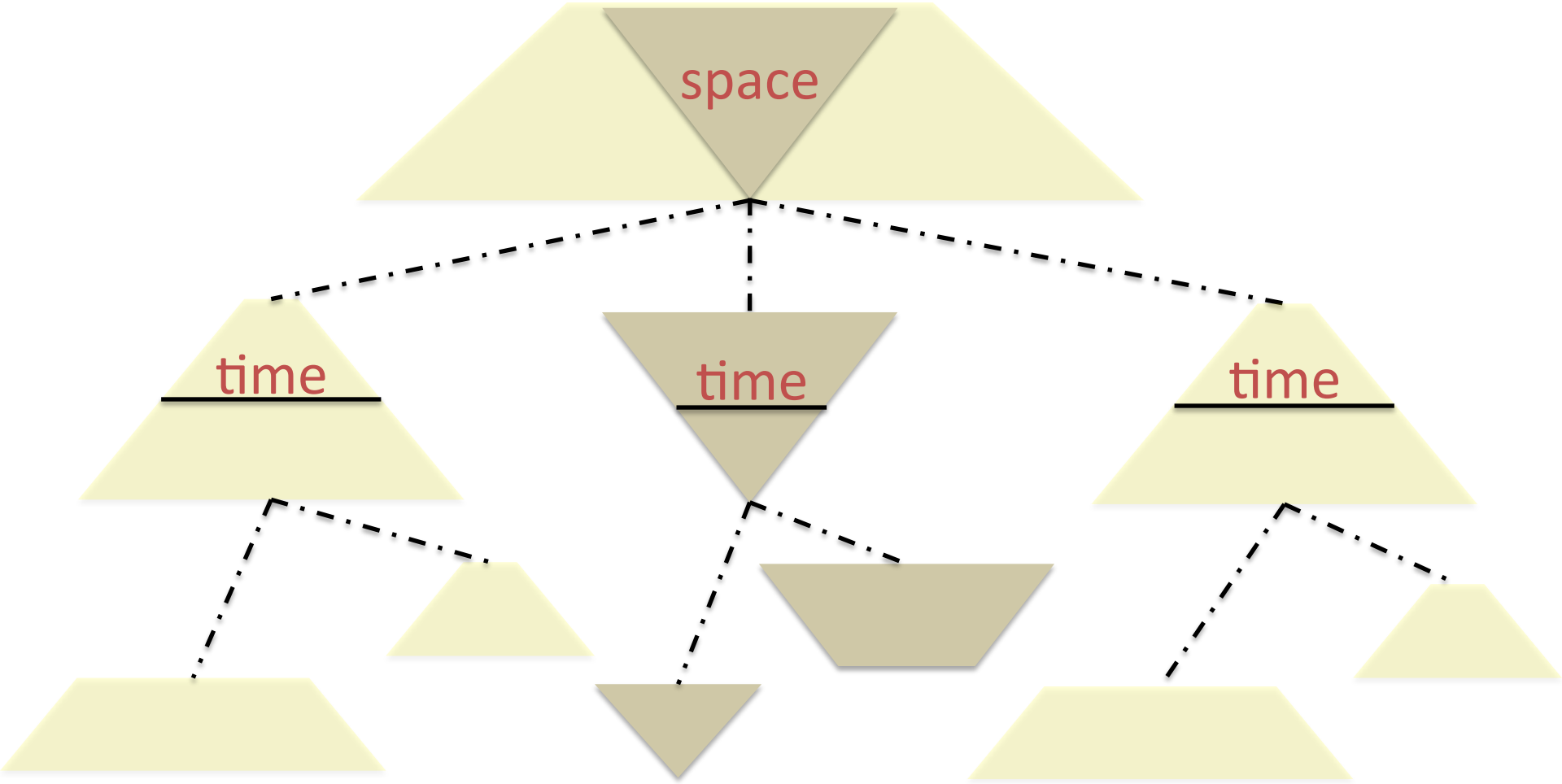
space

base case

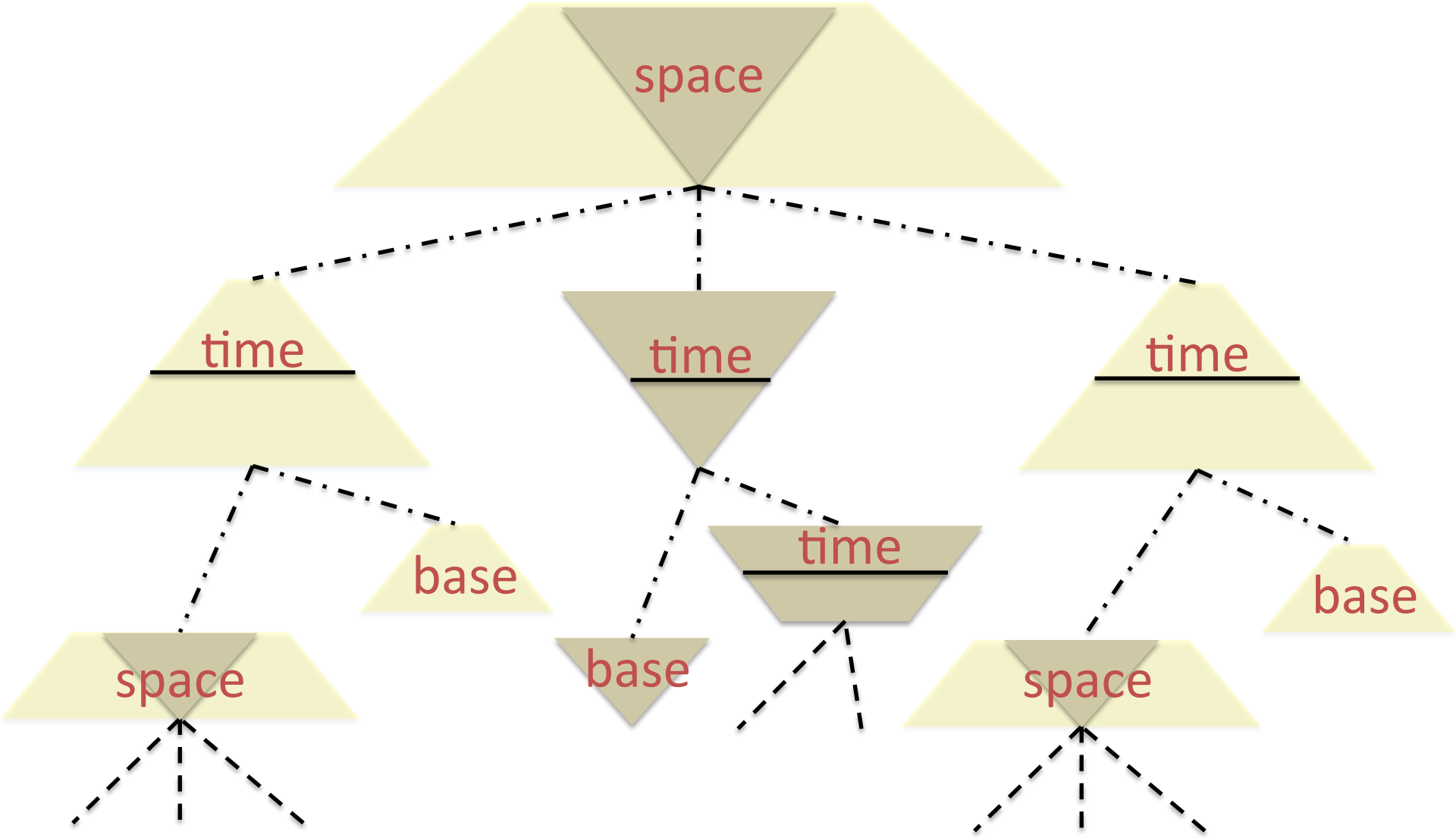
A TRAP divide-and-conquer tree



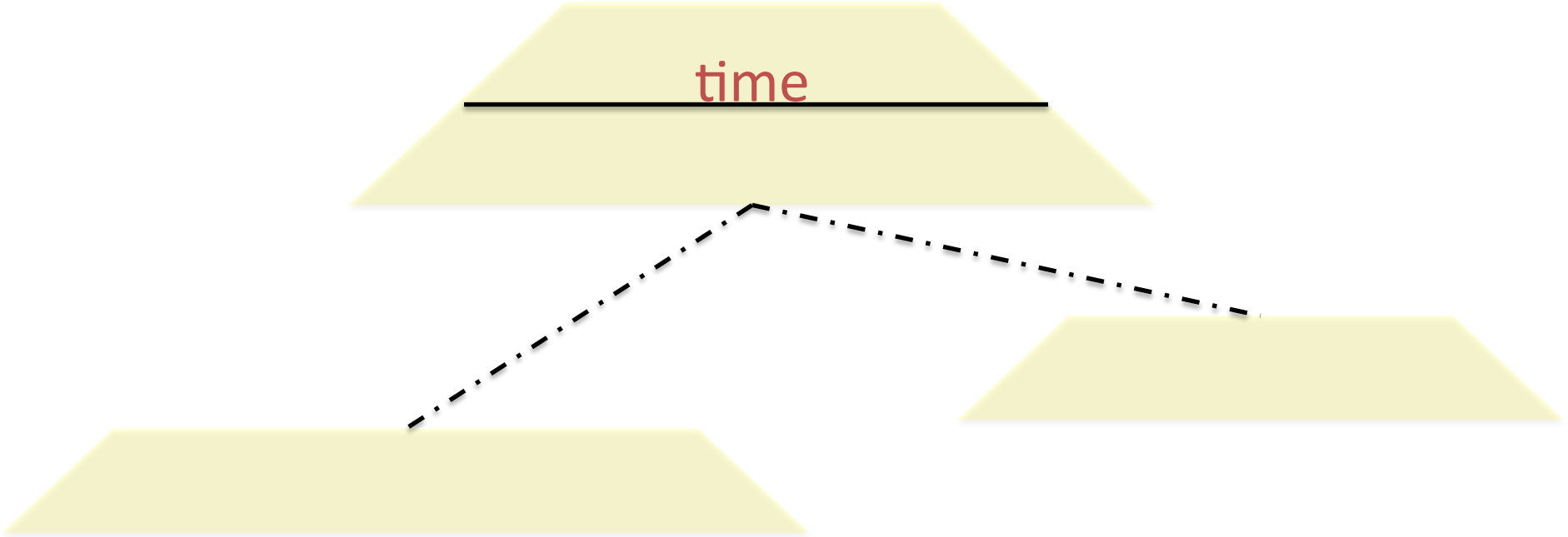
A TRAP divide-and-conquer tree



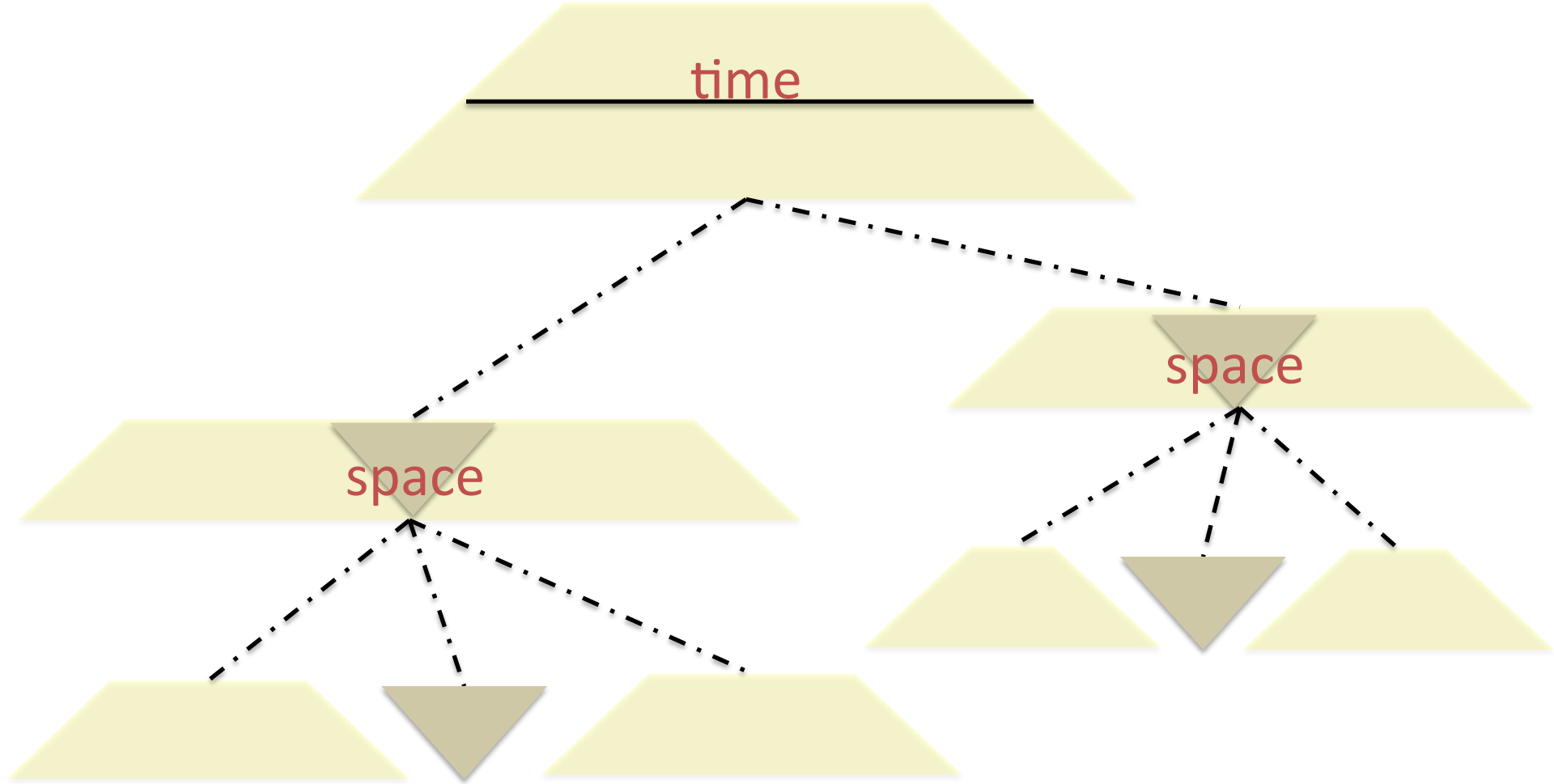
A TRAP divide-and-conquer tree



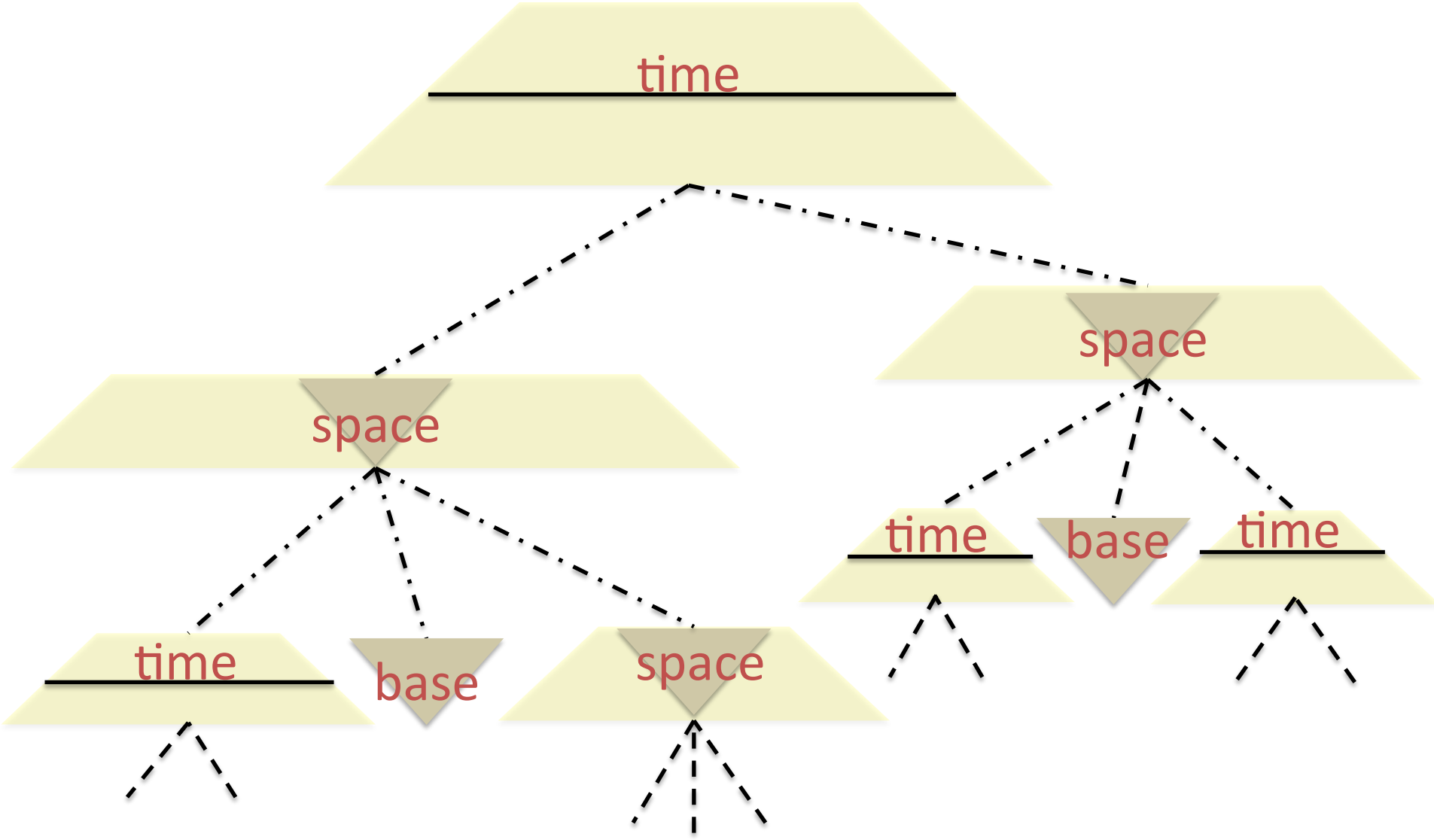
Another divide-and-conquer tree



Another divide-and-conquer tree



Another divide-and-conquer tree



Yet another divide-and-conquer tree

Autotuning strategy 2 : Search for optimal trees

Each tree might take a different time to execute. Find a tree with the fastest time.

Benefits :
Avoids the “one size fits all” problem in the base domain

The search domain of trees : “Choice domain”

The choice domain consists of the set of all possible trees for the given space-time grid.

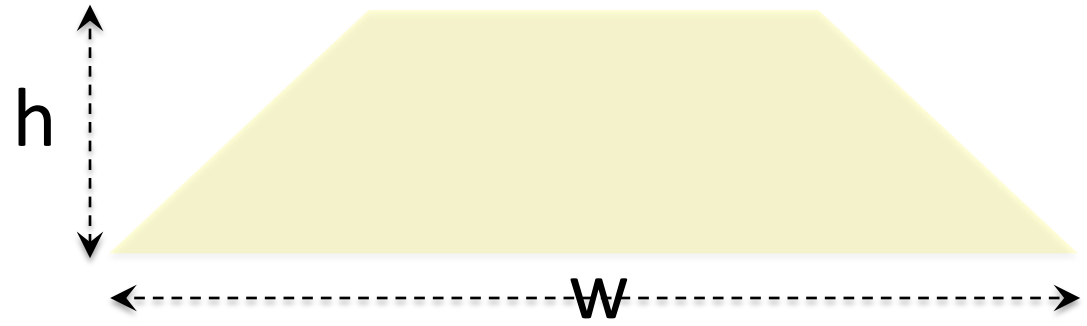
Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - Ztune and its pruning properties

Ztune

An exhaustive autotuner that searches the choice domain for a tree with the fastest runtime.

Problem 1 : The choice domain is large

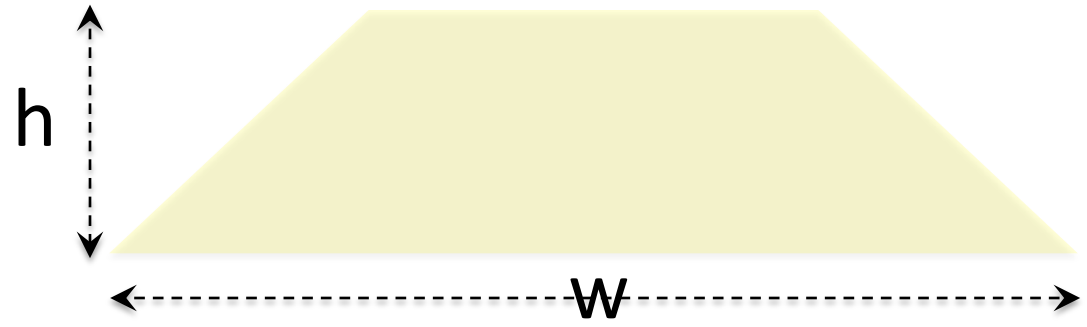


The number of trees is exponential
(at least $2^{h/2}$)
in the height of the trapezoid.

Exhaustive search is time consuming

Does not find optimal trees for any benchmark in 2 days.

Problem 2 : An optimal tree is too big to store



TRAP uses only $\Theta(w)$ memory

Memory overhead to store the tree is $\Omega(hw)$ (since the tree has $\Theta(hw)$ leaves).

Ztune's Pruning properties

1. Space-time equivalence [Frigo, Johnson 1999]
   - Uses dynamic programming to avoid finding plans for similar subproblems.
2. Divide subsumption
   - Learns from division choices of children to avoid finding "base-case costs" of parents.
3. Dimensional priority [Datta 2009]
   - Prunes search domain using the fact that computations along contiguous memory locations are faster.

Comparison of plan-finding times

Name	Dim	STE	STE + DS	STE + DS + DP
APOP	1	∞	35 s	35 s
Heat1	2	3 h	52 s	20 s
Heat2	2	23 h	9 m	7 m
Life	2	∞	8 m	4 m
Heat3	2	∞	4 m	1 m
Heat4	2	∞	2m	47 s
Heat5	2	∞	20 m	7 m
LBM	3	29 h	12 m	34 s
Wave	3	∞	15 h	52 m
Heat6	4	∞	9 h	6 m

Comparison of runtimes (in s)

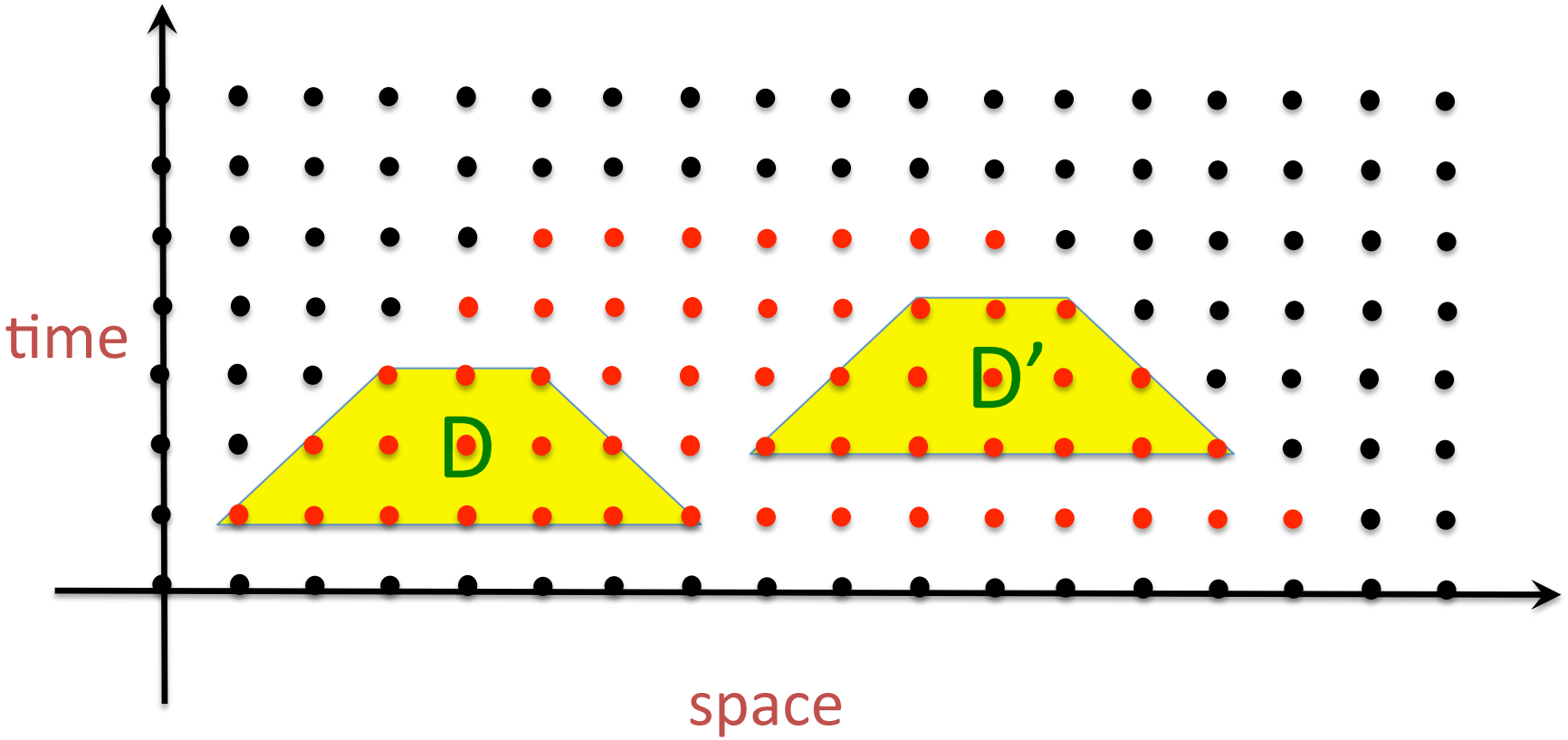
Name	STE	STE + DS	STE + DS + DP
APOP	--	1,248.84	1,246.98
Heat1	1.62	1.61	1.70
Heat2	50.24	50.01	50.77
Life	--	16.57	17.04
Heat3	--	25.01	24.75
Heat4	--	19.65	19.61
Heat5	--	575.07	559.01
LBM	13.45	13.40	13.24
Wave	--	826.19	815.12
Heat6	--	18.81	18.99

Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - Space-time equivalence

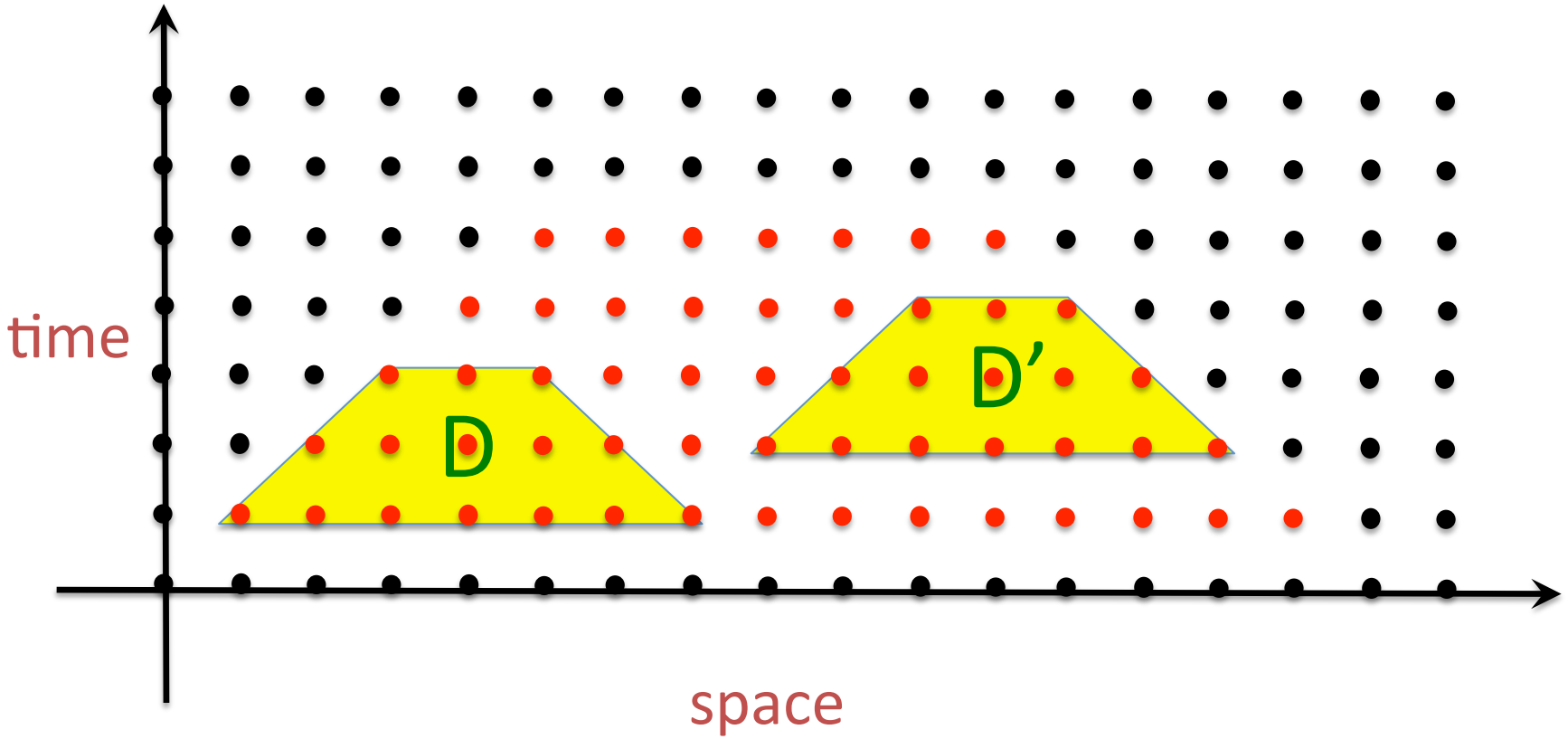
Don’t find plan for all nodes in the tree

D and D’ are space-time equivalent, if D is a translation of D’ in the space-time.



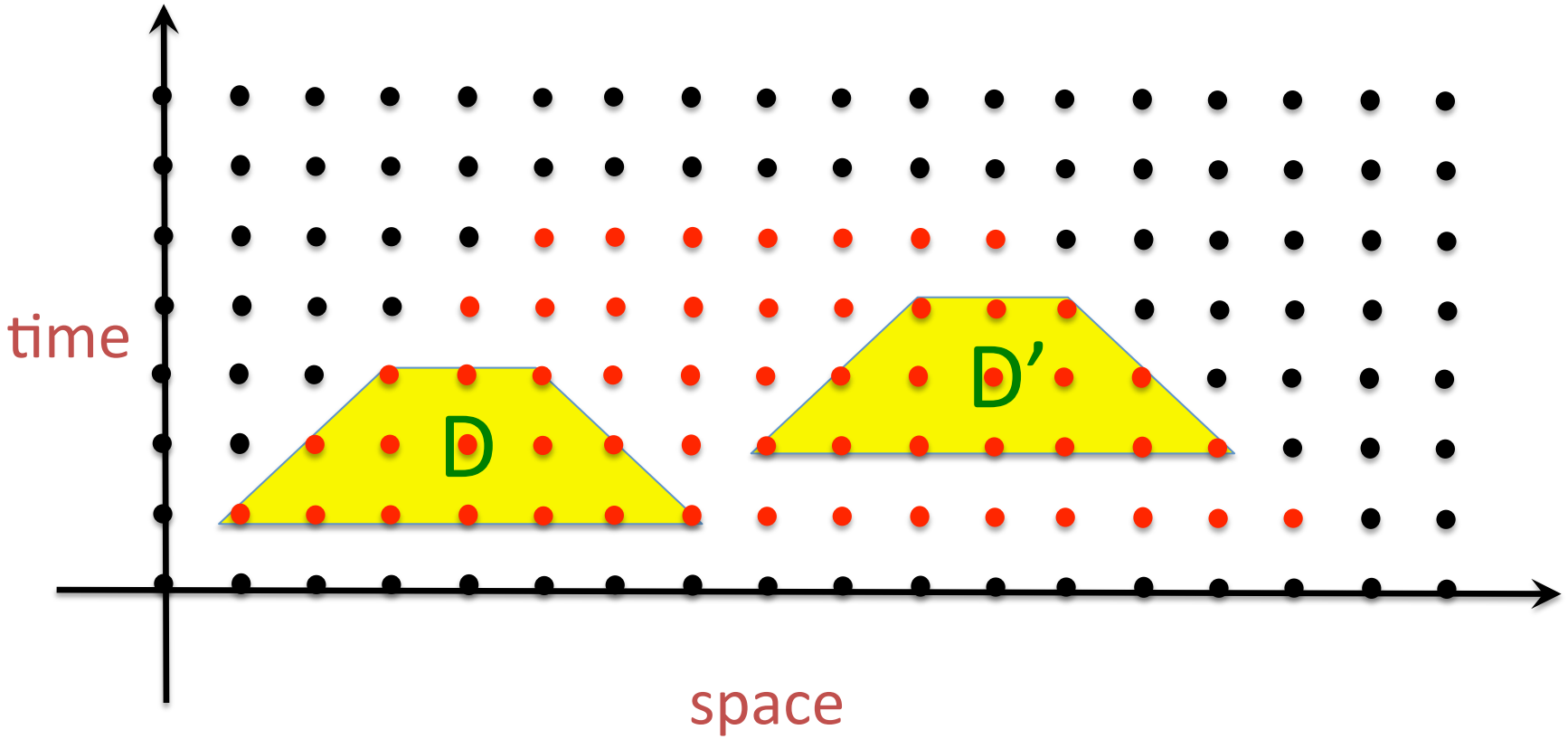
Space-time equivalence

Assume D and D’ have the same plan.
Find plan for either D or D’.



Memory overhead under space-time equivalence

Reduces memory overhead to O(w lgh)

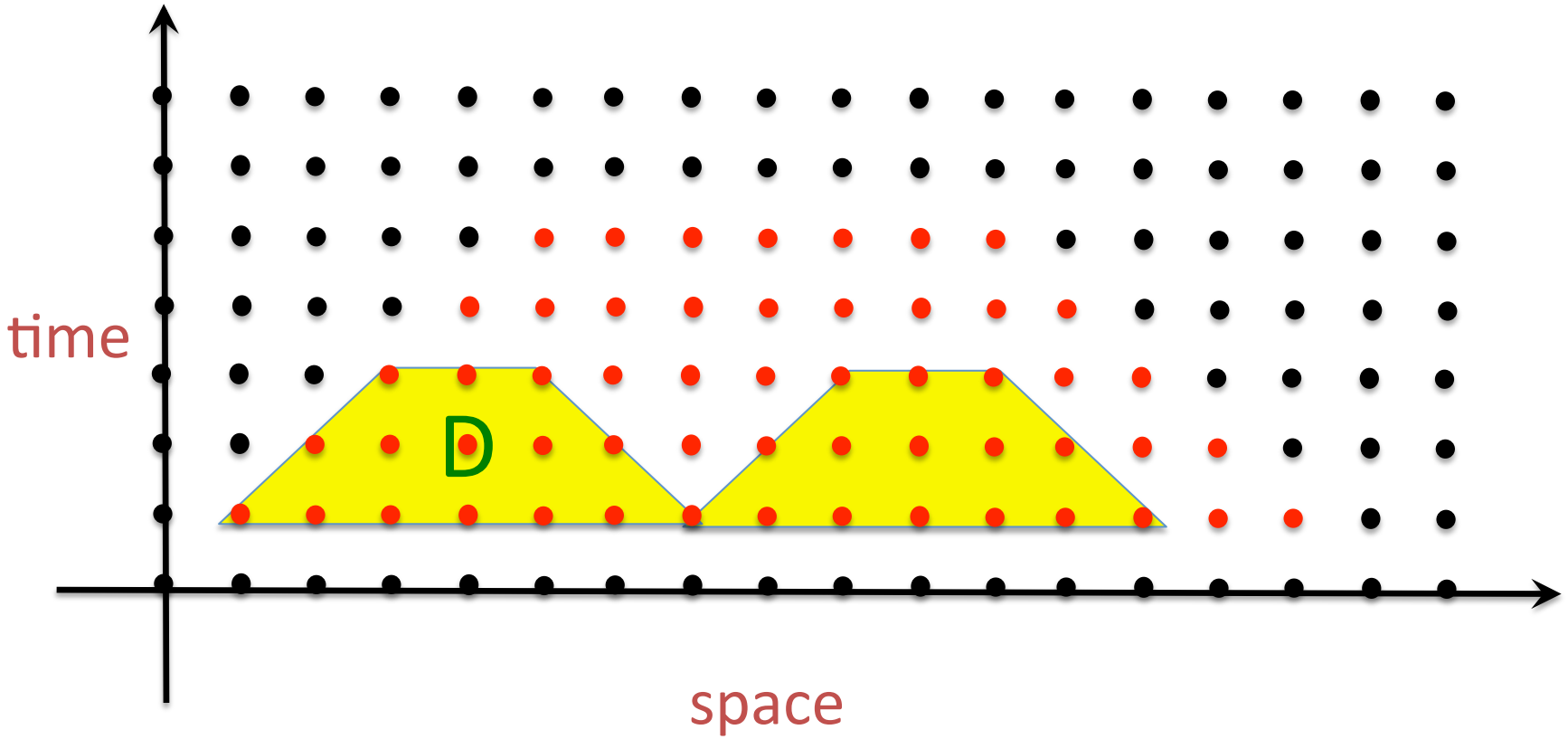


Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - Space-time equivalence
  - Divide subsumption

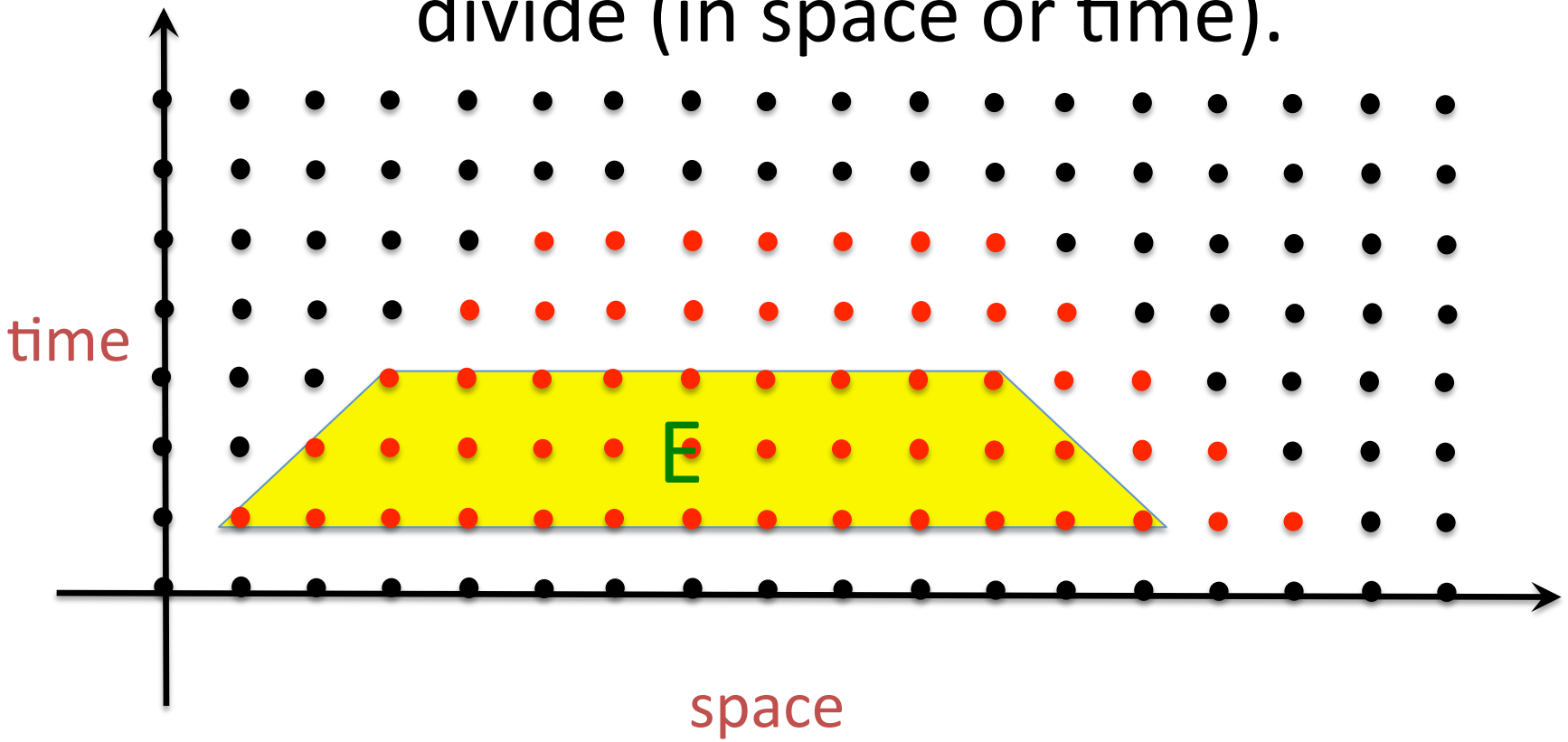
Learn from your child's experience

Suppose the optimal choice for D is to divide (in space or time).



Divide subsumption

Then the optimal choice for E, which is the parent of D in the tree, must be to divide (in space or time).



Why does divide subsumption hold?

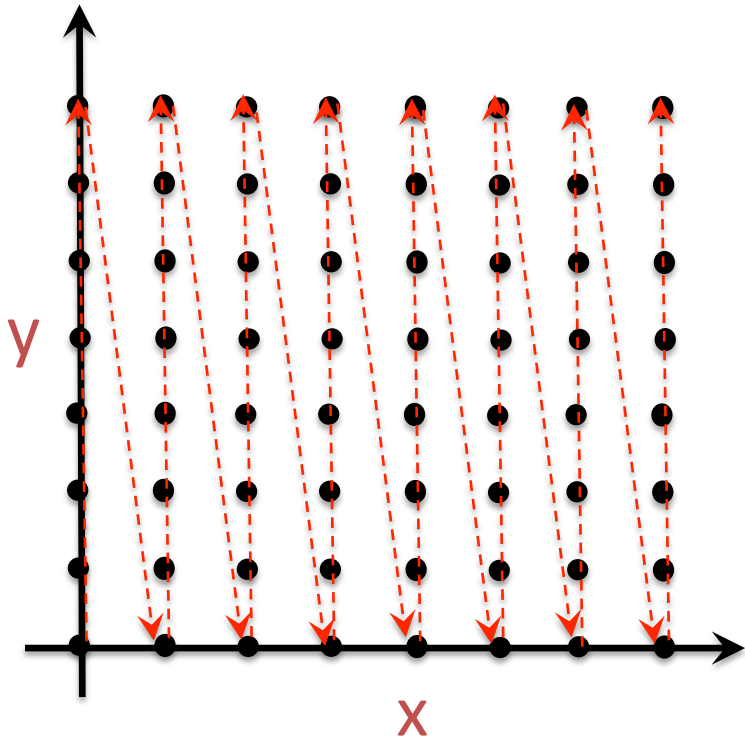
- D was divided since it didn't fit in cache and hence was expensive to coarsen as a base case.
- E is larger than D.
  - E will not fit in cache and will also be expensive to coarsen as a base case.
- Avoid executing the base-case of E.

Suboutline

- Review of Pochoir's TRAP stencil algorithm [SPAA 2011]
- Autotuning TRAP
  - Space-time equivalence
  - Divide subsumption
  - Dimensional priority

Exploit memory layout

- Consider a 2D XY spatial grid, stored in column major order.
- Computations along the contiguous y dimension are faster than those along the x dimension, due to vectorization, prefetching, and other techniques.



Dimensional priority

- Divide along at most 1 spatial dimension.
  - Choose any non-contiguous dimension.
  - If no non-contiguous dimension can be divided, choose the contiguous dimension.

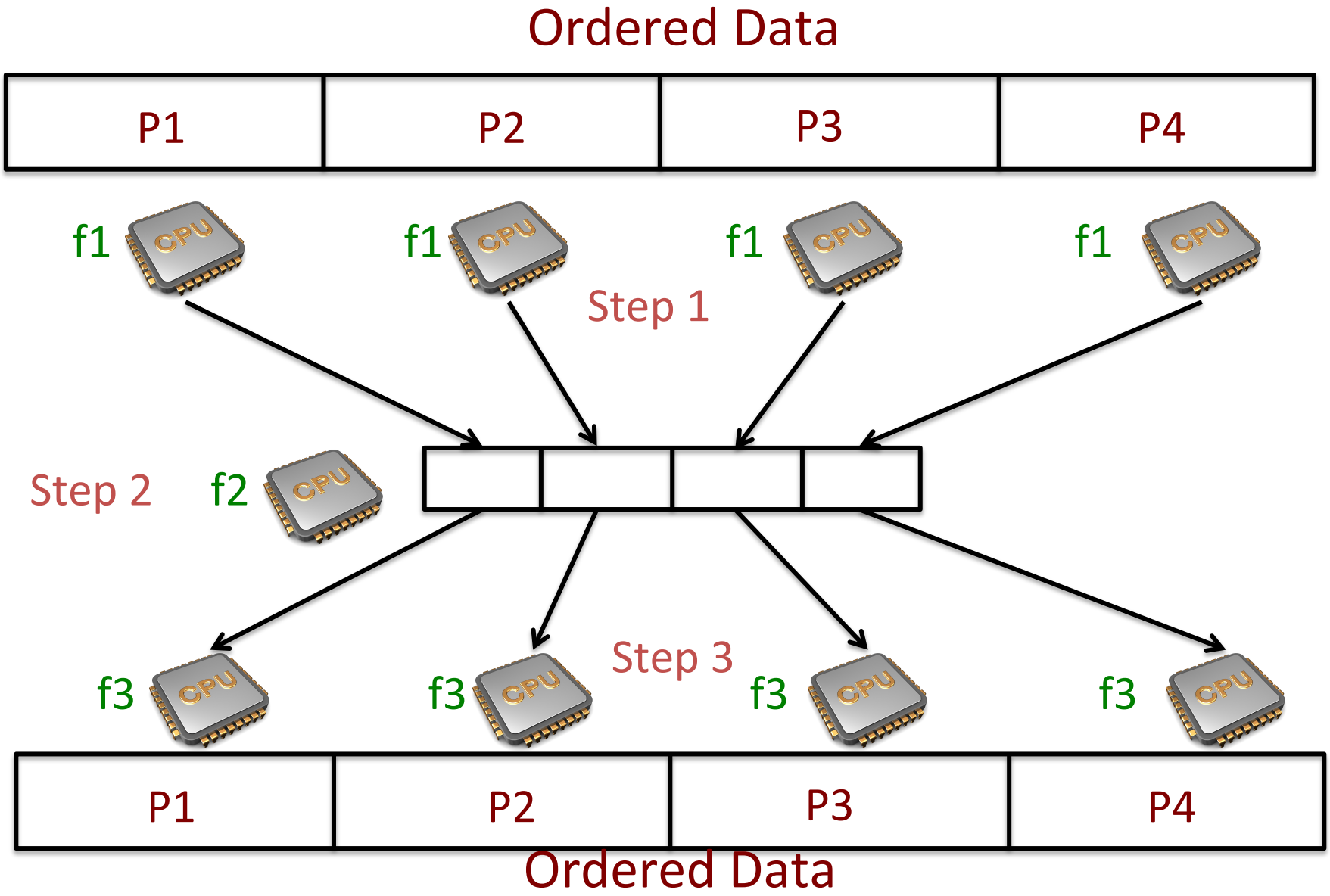
Can the choice domain be heuristically autotuned?

- Parameterizing the division choice at each recursive step creates enormous memory overhead.
  - For example, $\theta(wh)$ parameters for a 2D grid with width w and height h.
- A heuristic autotuner is unable to leverage the pruning properties since it treats the program being autotuned as a black box.

Outline

- Ztune
  - Autotune the stencil code in Pochoir stencil compiler.[SPAA 2011]
- Star
  - A “Wasp” algorithm for parallel watershed cuts
- Future work

Recall the Star structure



Can the MapReduce abstraction be used to execute the Star structure?

MapReduce[Dean, Ghemawat 2004] is a simple yet powerful abstraction for parallelizing computations.

Using MapReduce

Users express their computations as two functions : map and reduce.

The map function

The map function takes as input a key-value pair, and creates a list of intermediate key-value pairs

map : (k1,v1) $\longrightarrow$ list(k2,v2)

The reduce function

The reduce function takes as input an intermediate key and the list of values for the key, and merges the values to a possibly smaller set of values.

reduce : (k2,list(v2)) ⟶ list(v2)

Suitability of MapReduce

- MapReduce is suitable for computations over unordered data sets.
  - For example, word count, sum, min, max, mean etc.
- However, MapReduce is not well-suited for computations over ordered data sets.
  - For example, consider the all-prefix-sum program from the star class.

The all-prefix-sum program

Given an ordered set A = [a_1, a_2,…, a_n], the all-prefix-sum program returns the ordered set [a_1, a_1+ a_2,…, a_1+ a_2+…+a_n].

For example, given A = [10, 20, 30, 40]
the output is the ordered set [10, 30, 60, 100].

Takes θ(n) time serial time.
Star solves all-prefix-sum in θ(n/p + p) time and θ(p) communication, given p processors.

Express all-prefix-sum in MapReduce

For $1 \leq i \leq n$, define the map and reduce functions as follows :

map : $(i, a_i) \longrightarrow [(i, a_i), (i+1, a_i), \ldots, (n, a_i)]$

reduce : $(i, [a_1, a_2, \ldots, a_i]) \longrightarrow (i, a_1 + a_2 + \ldots + a_i)$

Takes $\theta(n^2)$ work and communication, and $\theta(n)$ time.

A better all-prefix-sum in MapReduce

For $1 \le i \le n$, define the map function

map : $(i, a_i) \longrightarrow (1, (i, a_i))$

The reduce function

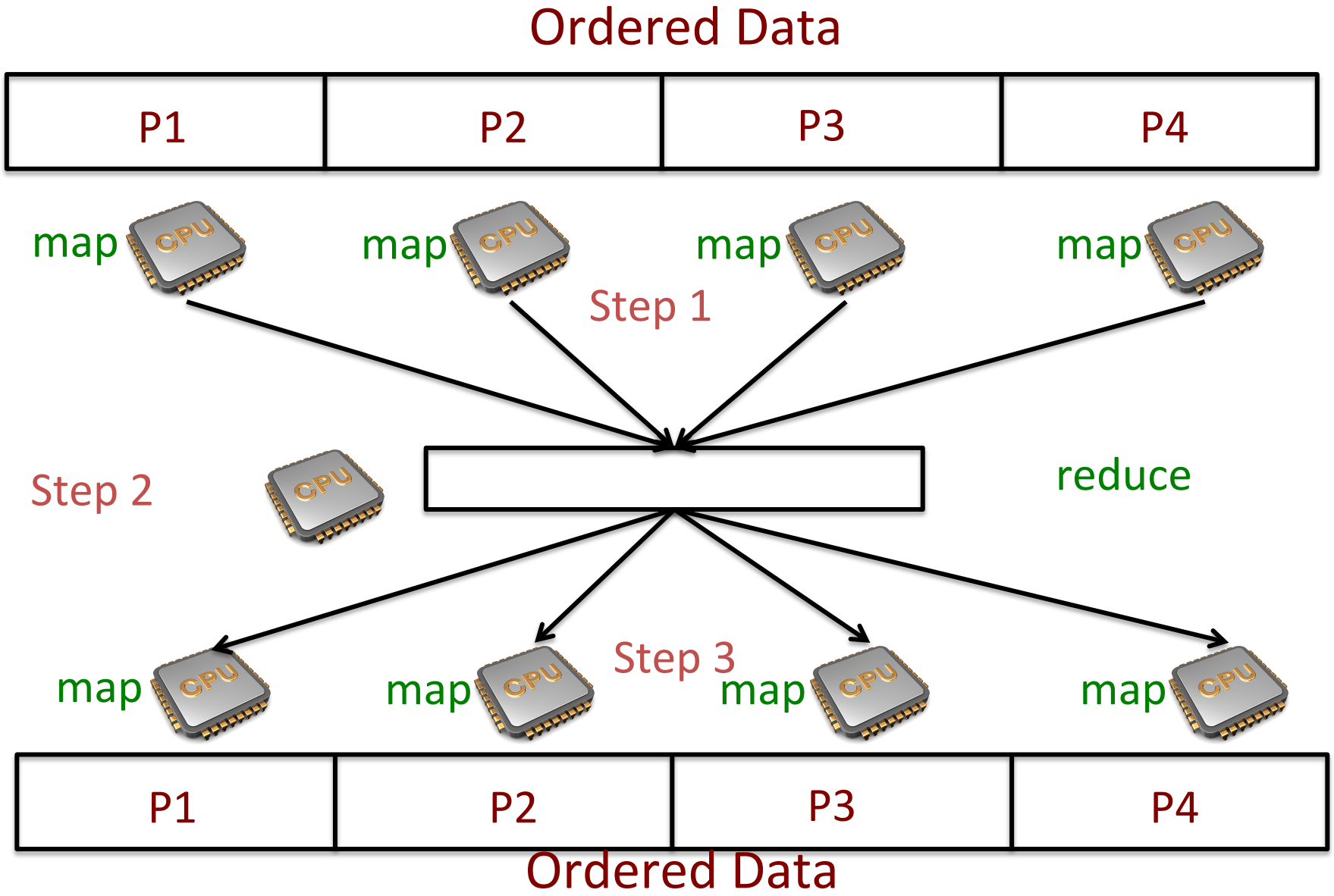
- receives the unsorted list of (i, a_i) pairs
- sorts the list in the increasing order of the indexes i, and
- computes all-prefix-sum on the sorted list

Takes $\theta(n \lg n)$ time using a standard sorting algorithm, and $\theta(n)$ communication.

The 3 subcomputations for all-prefix-sums in Star

1. Sum the elements in each partition in parallel
2. Compute an "exclusive prefix-sum" in serial
   - Given an ordered set A = [a_1, a_2,…, a_n], returns the ordered set [0, a_1,…, a_1+ a_2+…+a_{n-1}].
3. Compute all-prefix-sums in parallel using the output of step2 as an initial value.

Execute the Star structure in MapReduce



The optimized all-prefix-sum in MapReduce still takes longer

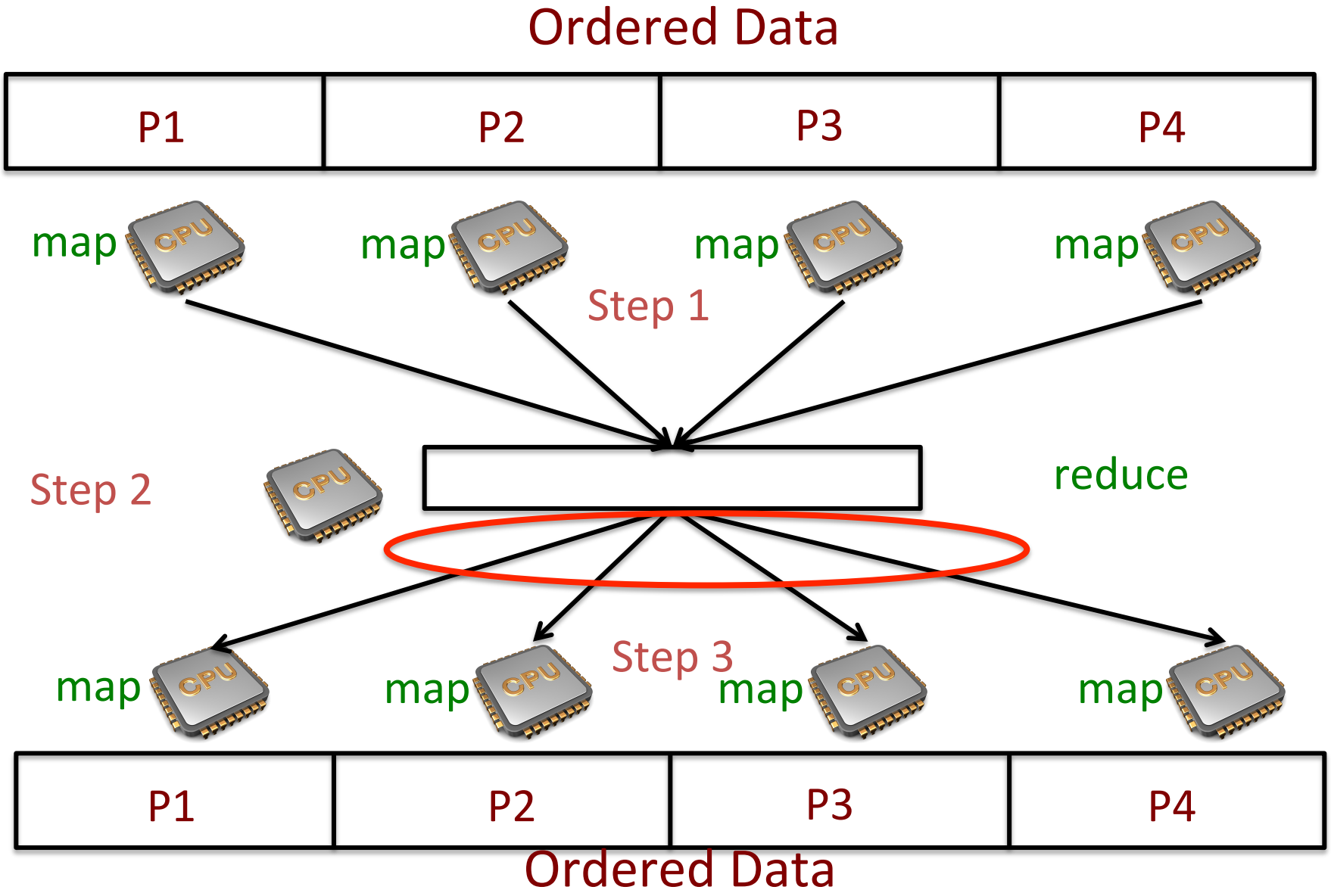
Given an ordered set of n elements and p processors, an optimized all-prefix-sum program in MapReduce still takes O(n/p lg n/p + p lg p) time, and $O(p^2)$ communication.

The fastest runtime occurs when p = sqrt(n), but communication increases to O(n).

Why is MapReduce having problems?

- MapReduce is not a batch processing framework
  - The map function operates on key-value pairs, and not on the whole partition
- MapReduce doesn't have a notion of order of data or partitions
  - The key-value pairs are shuffled, and have to be sorted to recreate the order.
- Communication between the second and third steps is "all-to-all".

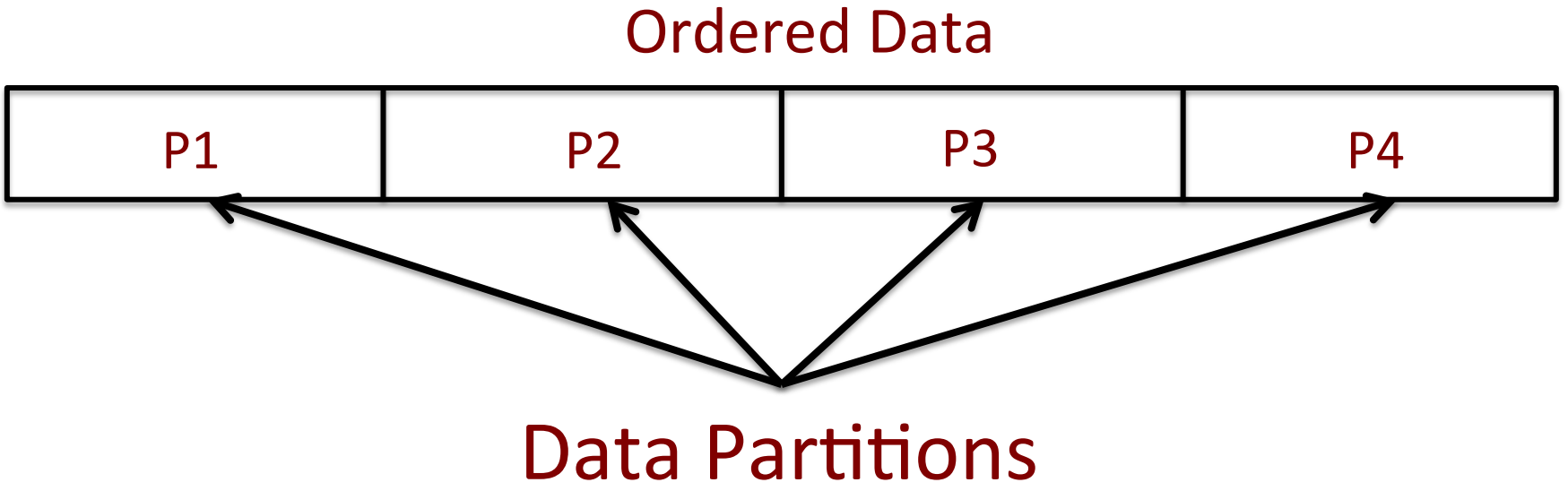
The all-to-all communication in MapReduce



How does Star avoid these problems?

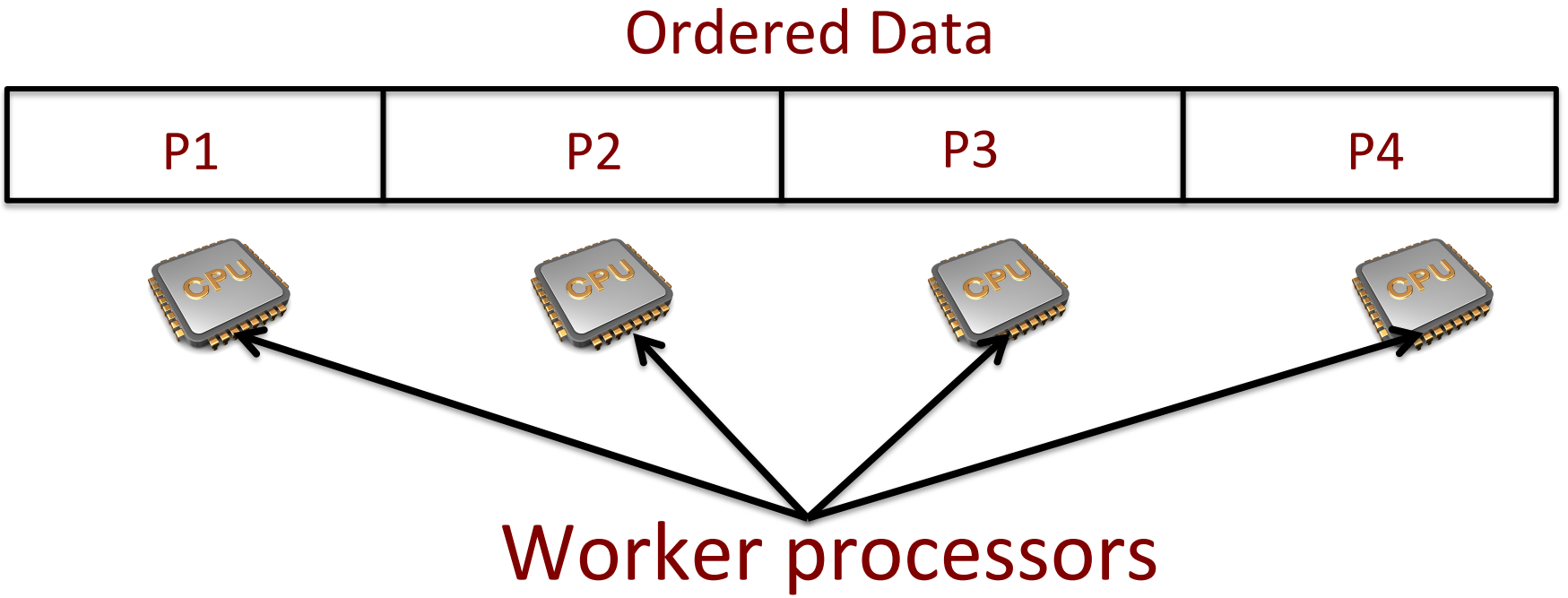
- Star is a batch processing framework, and operates on the whole partition
- Star is aware of the order of partitions

Execution of the subcomputations in Star

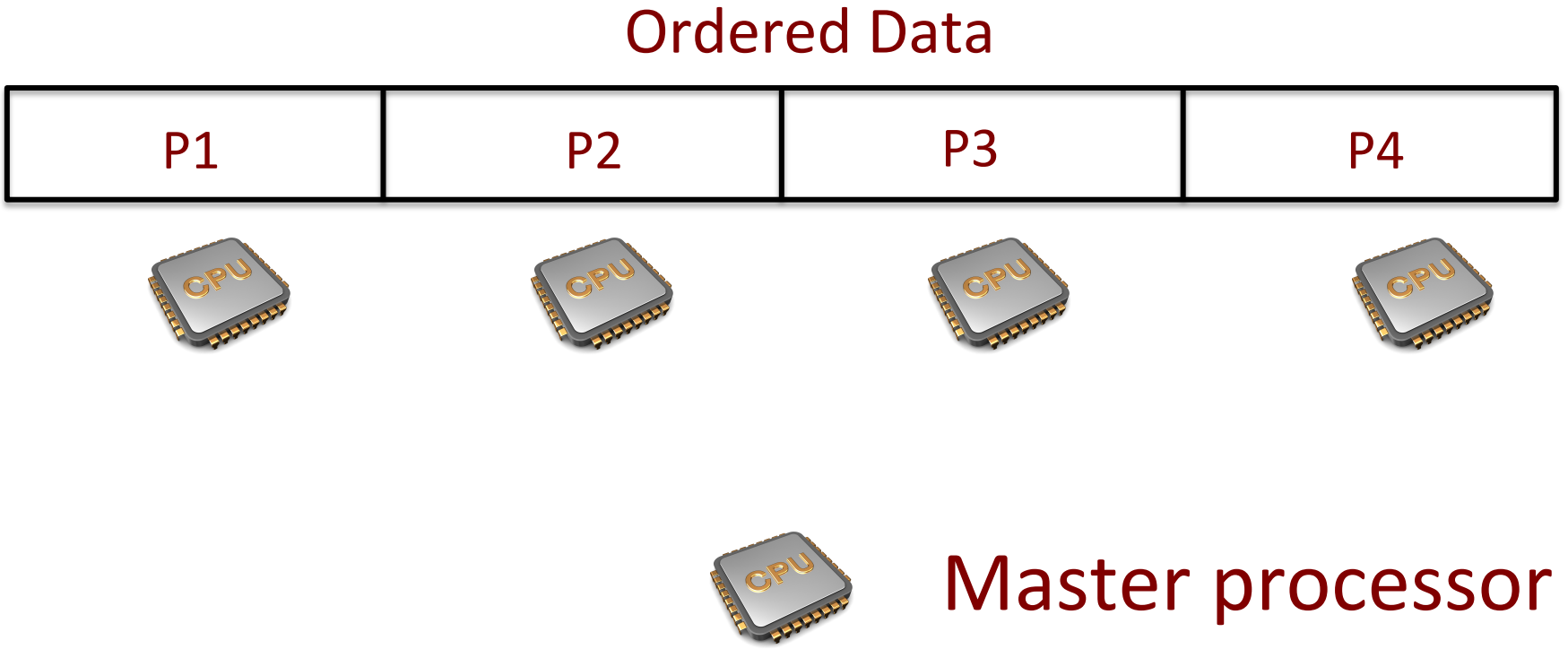


Arrange the ordered data in partitions

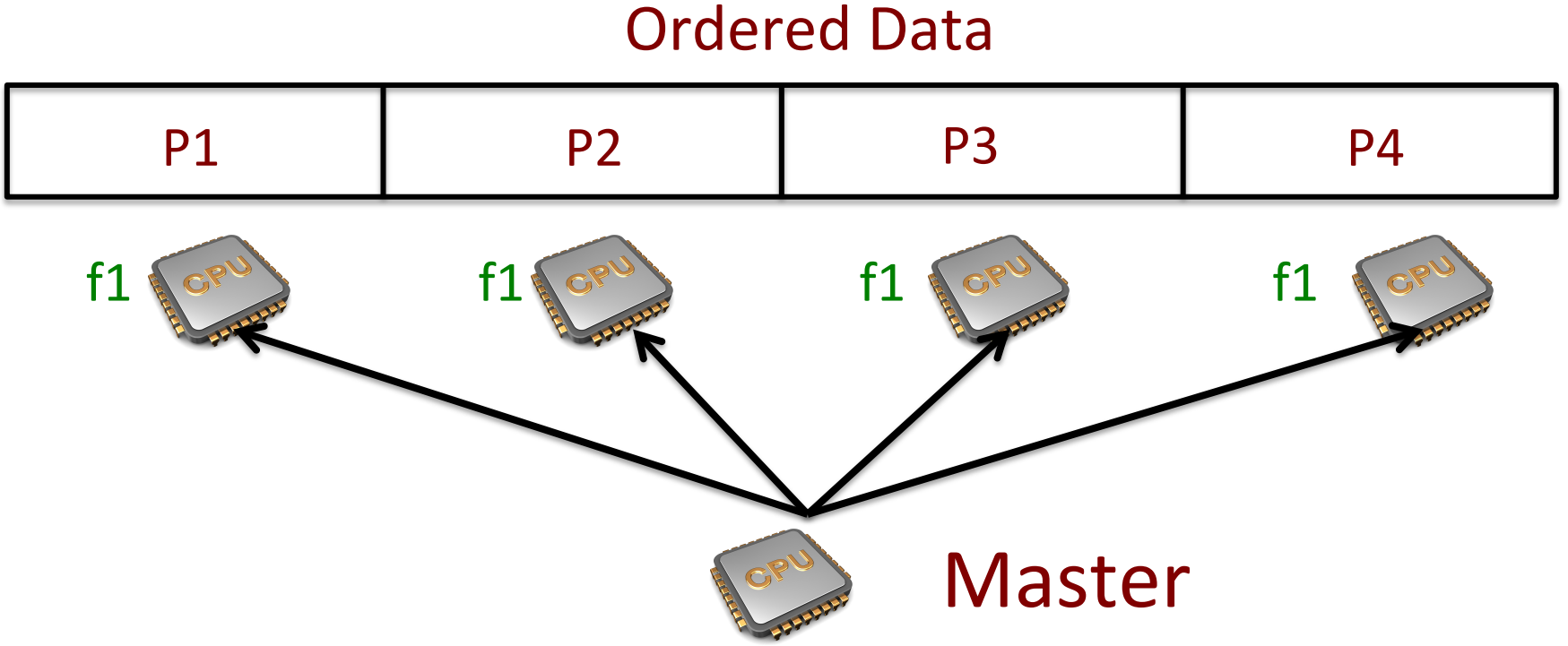
A processor is local to each partition



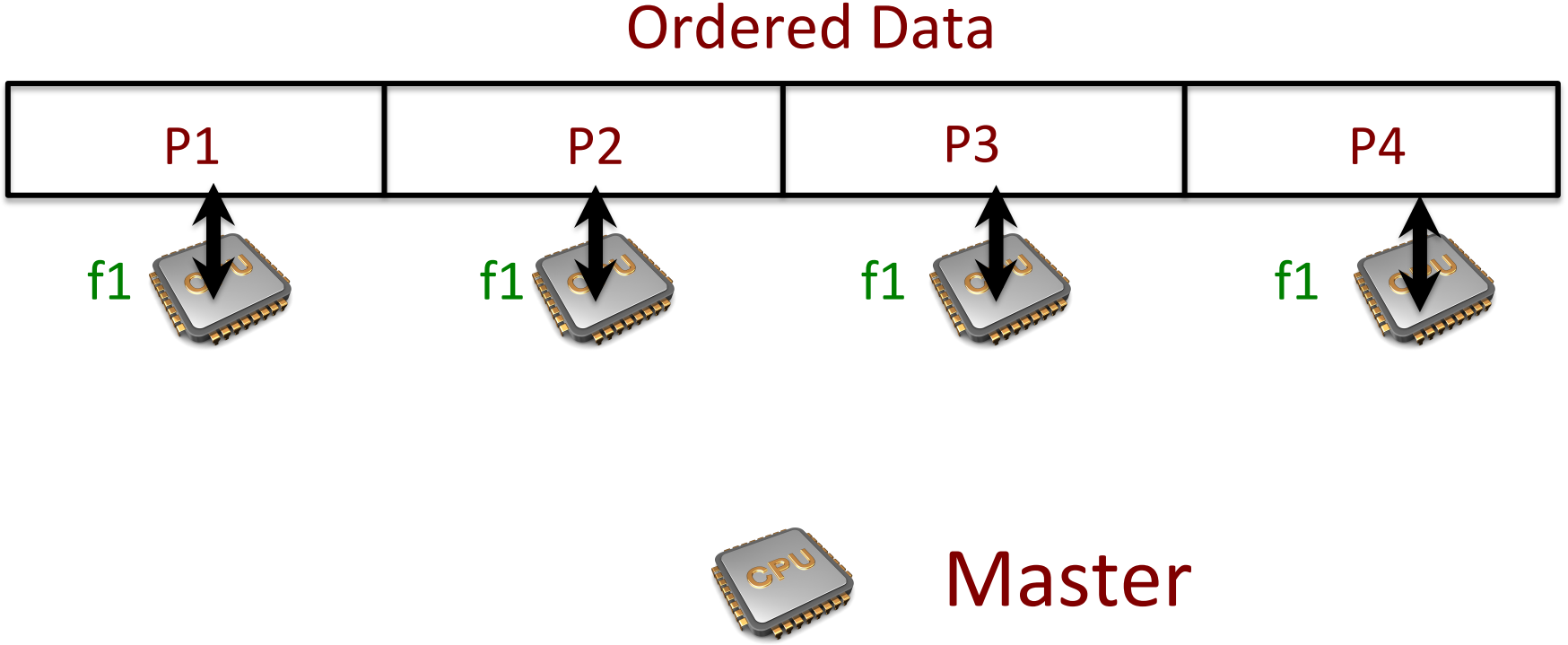
A Master controls the execution



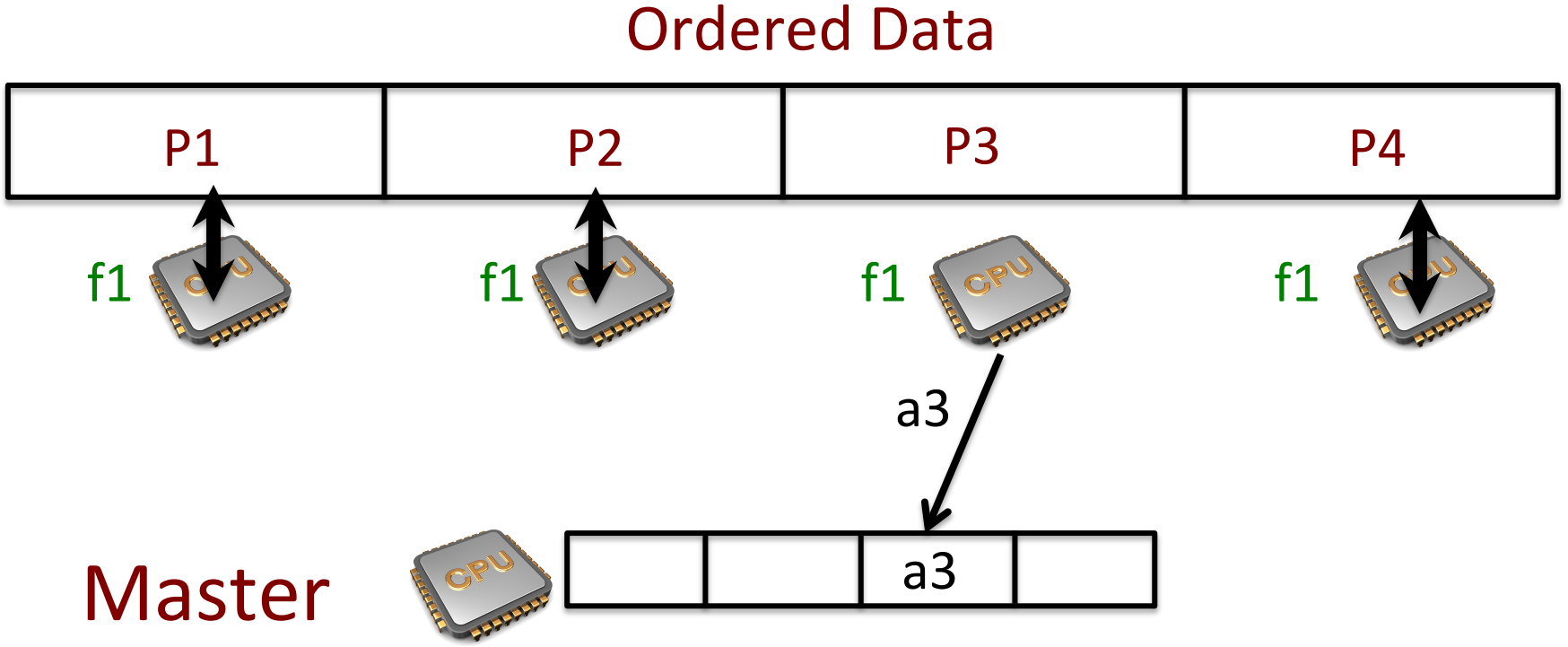
Step 1 : Master invokes the 1st subcomputation (f1) on the workers



Workers execute the 1st subcomputation (f1) in parallel

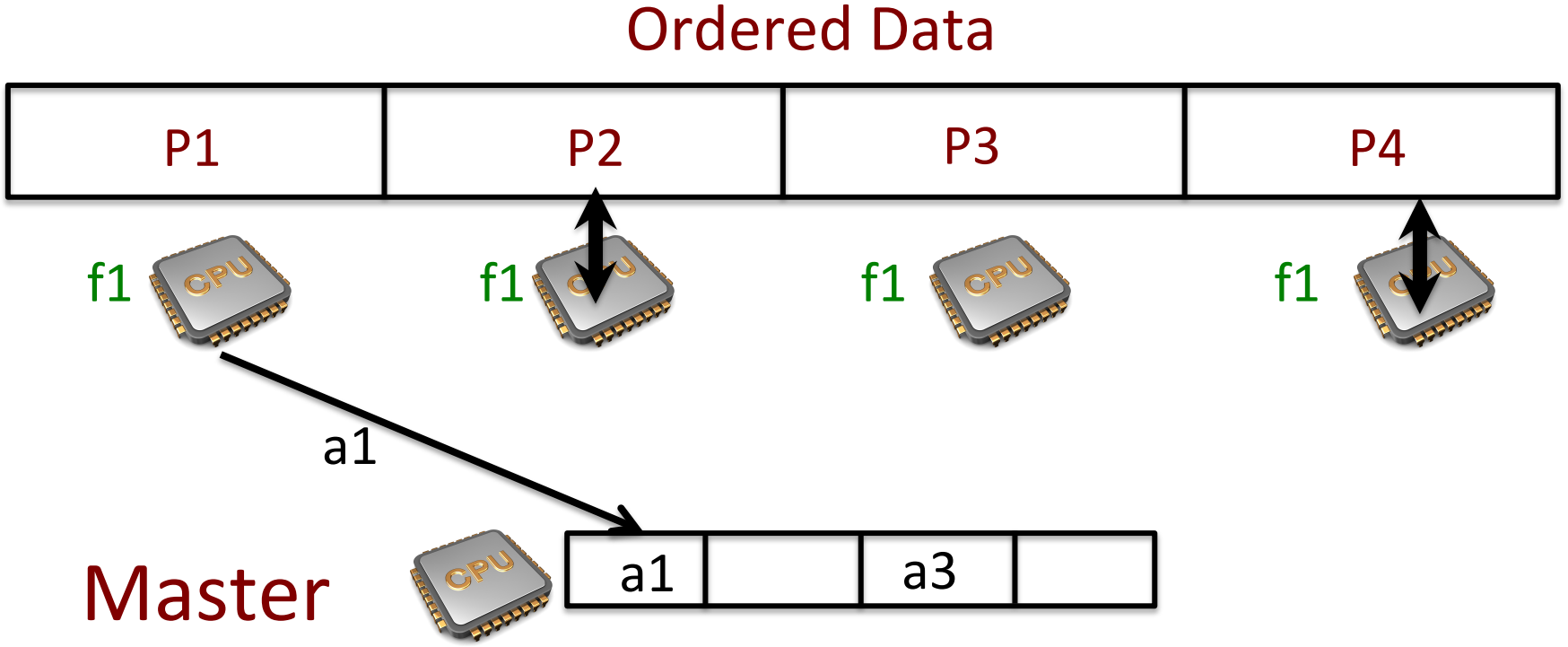


Master collects results from each worker



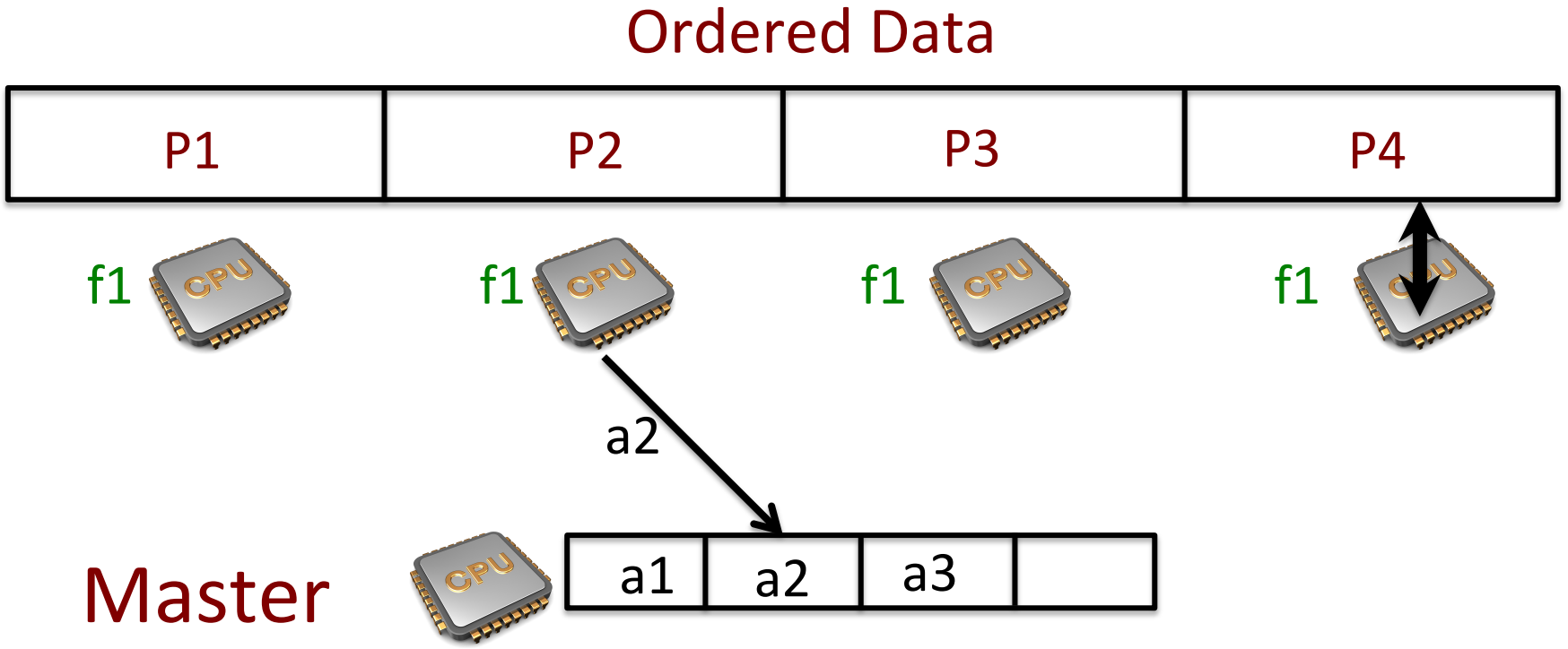
Master assembles the results from workers in the order of the partitions

Master collects results from each worker



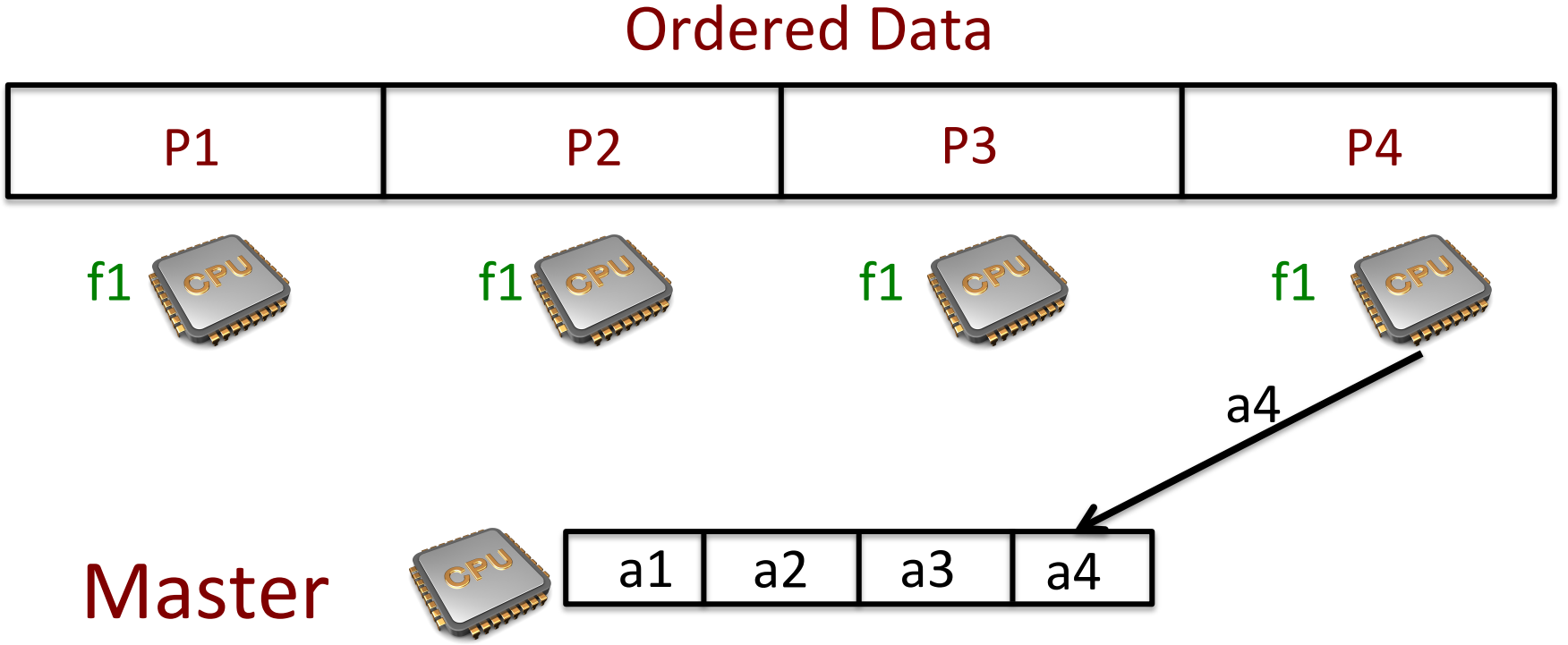
Master assembles the results from workers in the order of the partitions

Master collects results from each worker



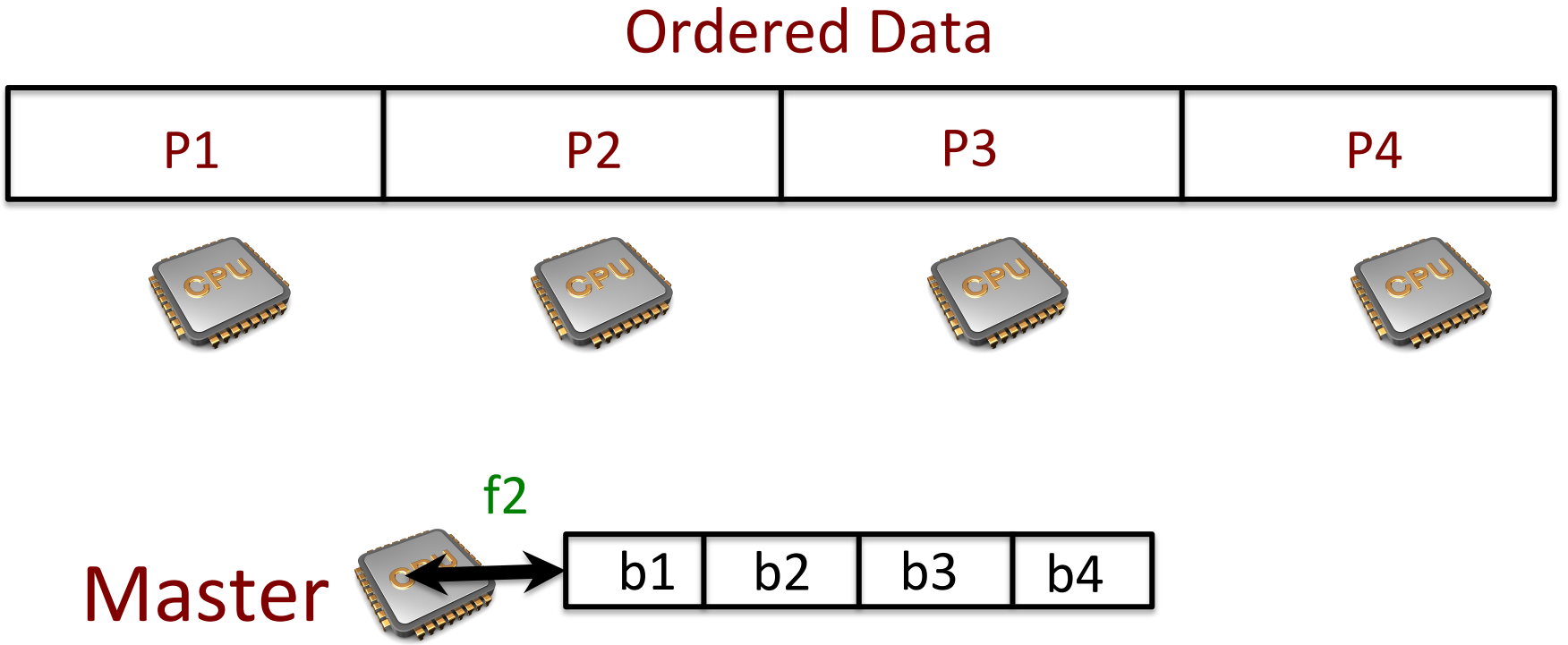
Master assembles the results from workers in the order of the partitions

Master collects results from each worker

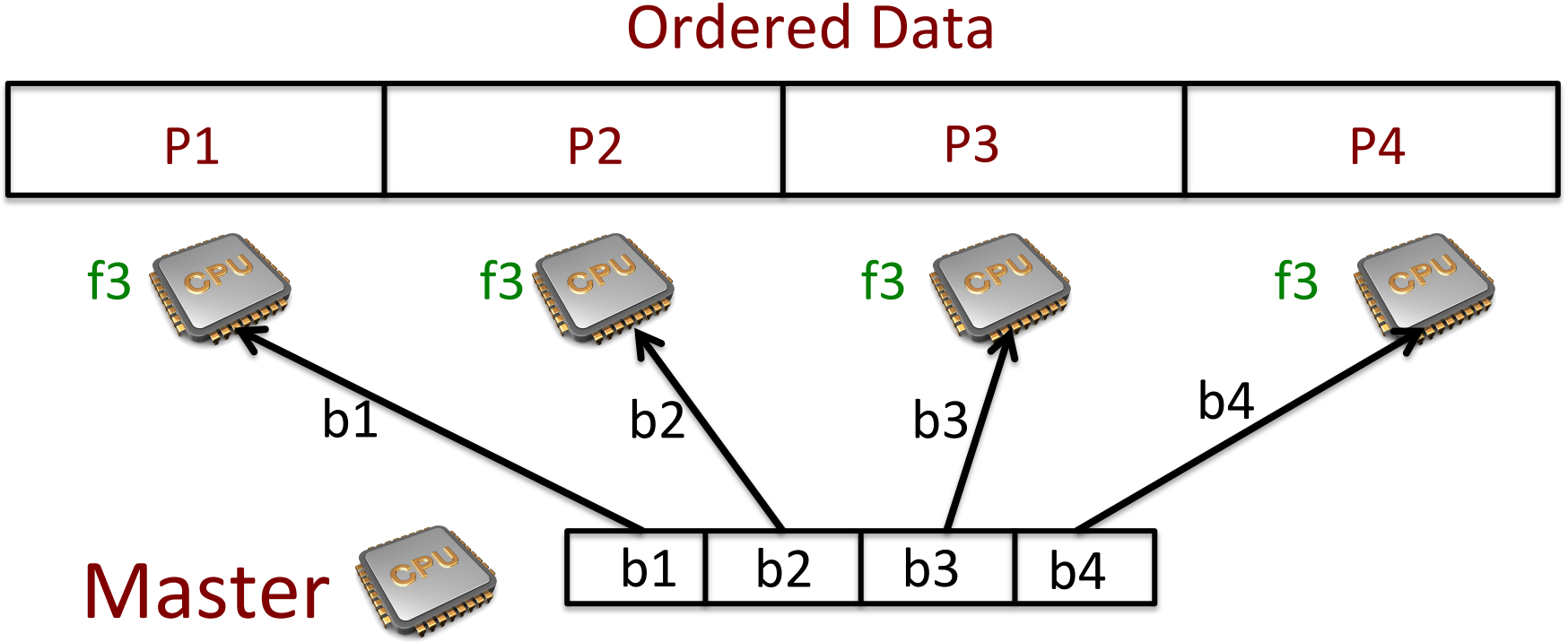


Master assembles the results from workers in the order of the partitions

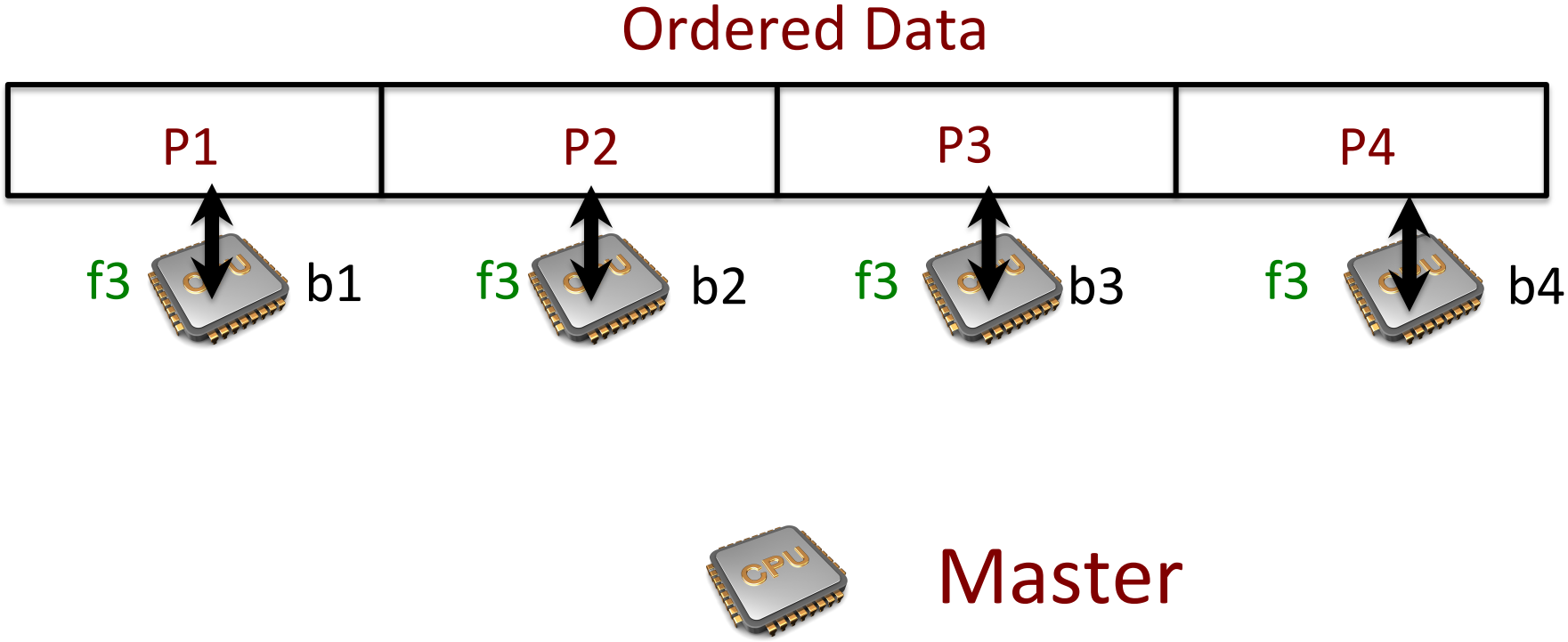
Step 2 : Master executes a serial noncommutative computation (f2)



Step 3 : Master invokes the 3rd subcomputation (f3) on the workers



Workers execute the 3rd subcomputation (f3) in parallel



The Star code written in Julia

Less than 30 lines of code in Julia

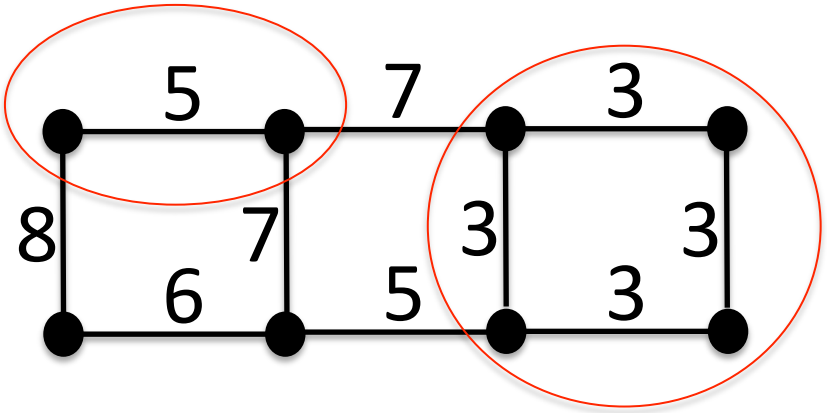
Leverages Julia's key capabilities :

- Concise expression of parallelism in code
- Support shared-memory and distributed memory parallel programs alike

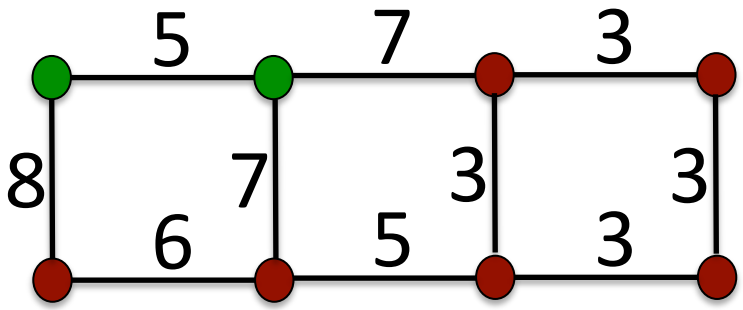
The Watershed problem

Given an edge-weighted graph G with a set M of “minima”, label each vertex v in G with a minimum m, such that there exists a steepest descending path from v to m.

An input edge-weighted graph with 2 minima



The watershed output



Prior art

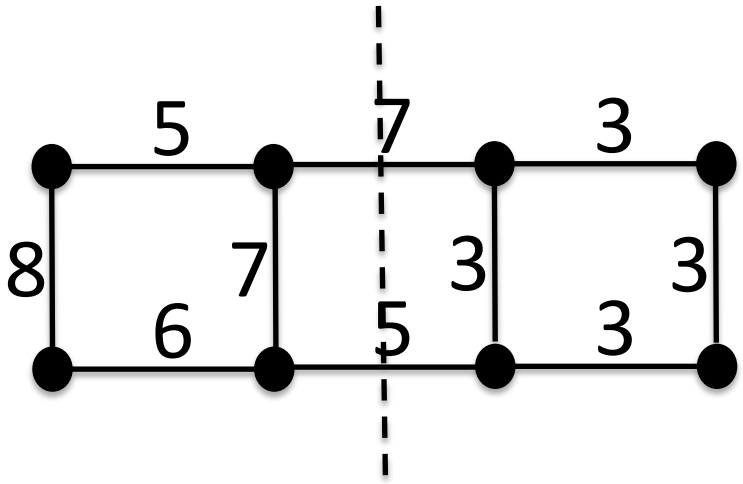
Cousty et al., 2009 provide a serial algorithm that runs in time proportional to the number of edges in graph.

Zlateski 2011, offers a parallel out-of-core algorithm that runs in 2 steps, and doesn't provide bounds

Contribution : The Wasp watershed algorithm

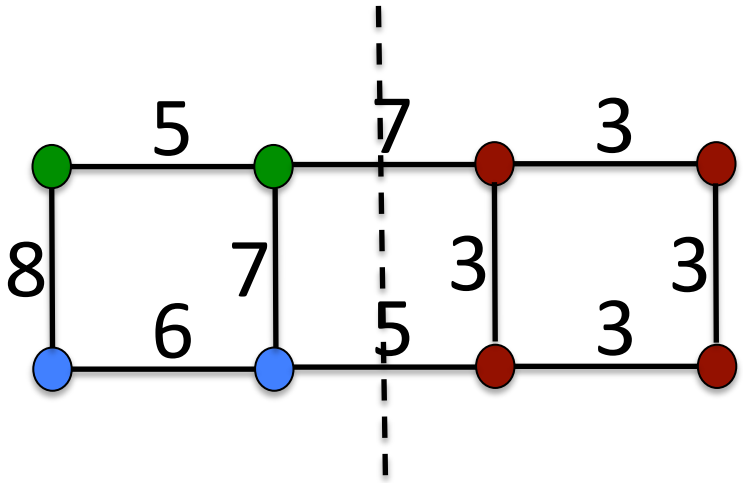
Given an square image with n pixels, which can be modeled as an edge-weighted graph, Wasp achieves $\theta(n/p + \sqrt{np}\ \alpha(\sqrt{np}))$ time and $\theta(\sqrt{np})$ communication using p processors, where α is the inverse Ackermann function.

Parallel watershed on 2 processors

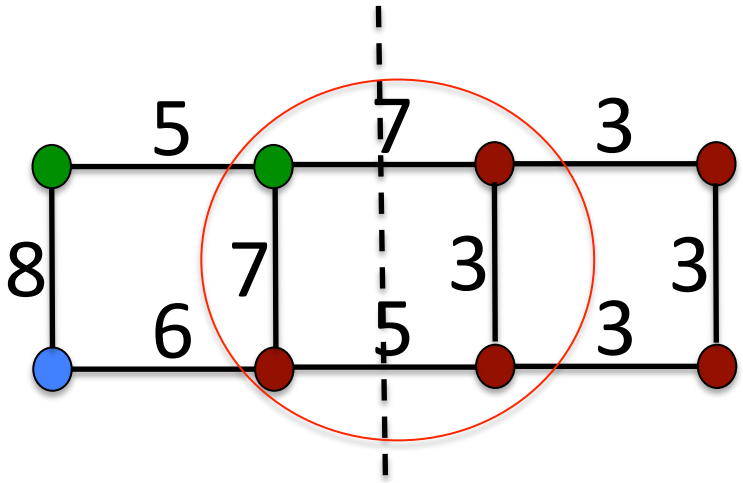


Partition the graph into 2 subgraphs.

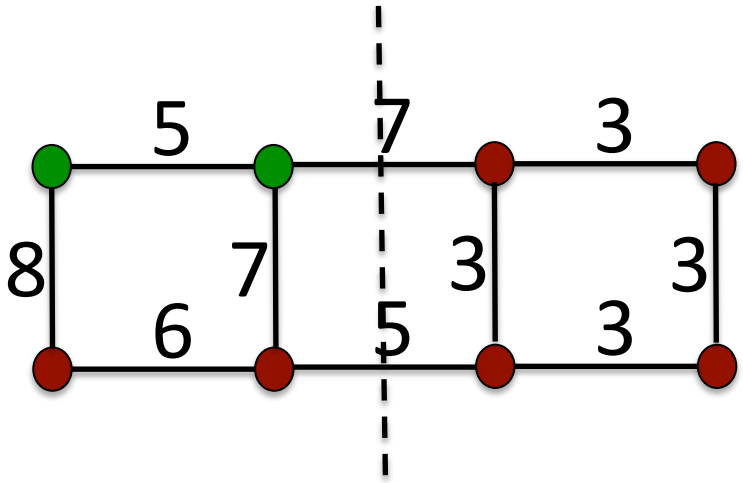
Step 1 : Run watershed on subgraphs in parallel



Step 2 : Reconcile the boundaries



Step 3 : Update the labels of the subgraphs in parallel



Parallel results for a 16384 X 16384 grid

# of processors	Runtime, s
1	581
2	334
4	186
8	102
16	51
32	28
64	15

Future work : Ztune

- Ztune other divide-and-conquer algorithms
  - Convolution, Dynamic programming.
- Bottom-up tuning
  - Autotune incrementally small problems. Use the results to tune big problems fast.
  - Example : Divide subsumption
- Parallel divide-and-conquer algorithms
  - Augment Ztune
  - New Challenges : Optimizing memory bandwidth usage

Future work : Star

- Fault tolerance, and load balancing
- Abstractions for recursive parallel computations

Questions?

Thank you