Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes parallel programming difficult and how to compare and judge programming models that aim to alleviate these difficulties

Presentation by: Andy Stone

April 27, 2010

Research Exam

Colorado State University

Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes **parallel programming** difficult and how to compare and judge programming models that aim to alleviate these difficulties

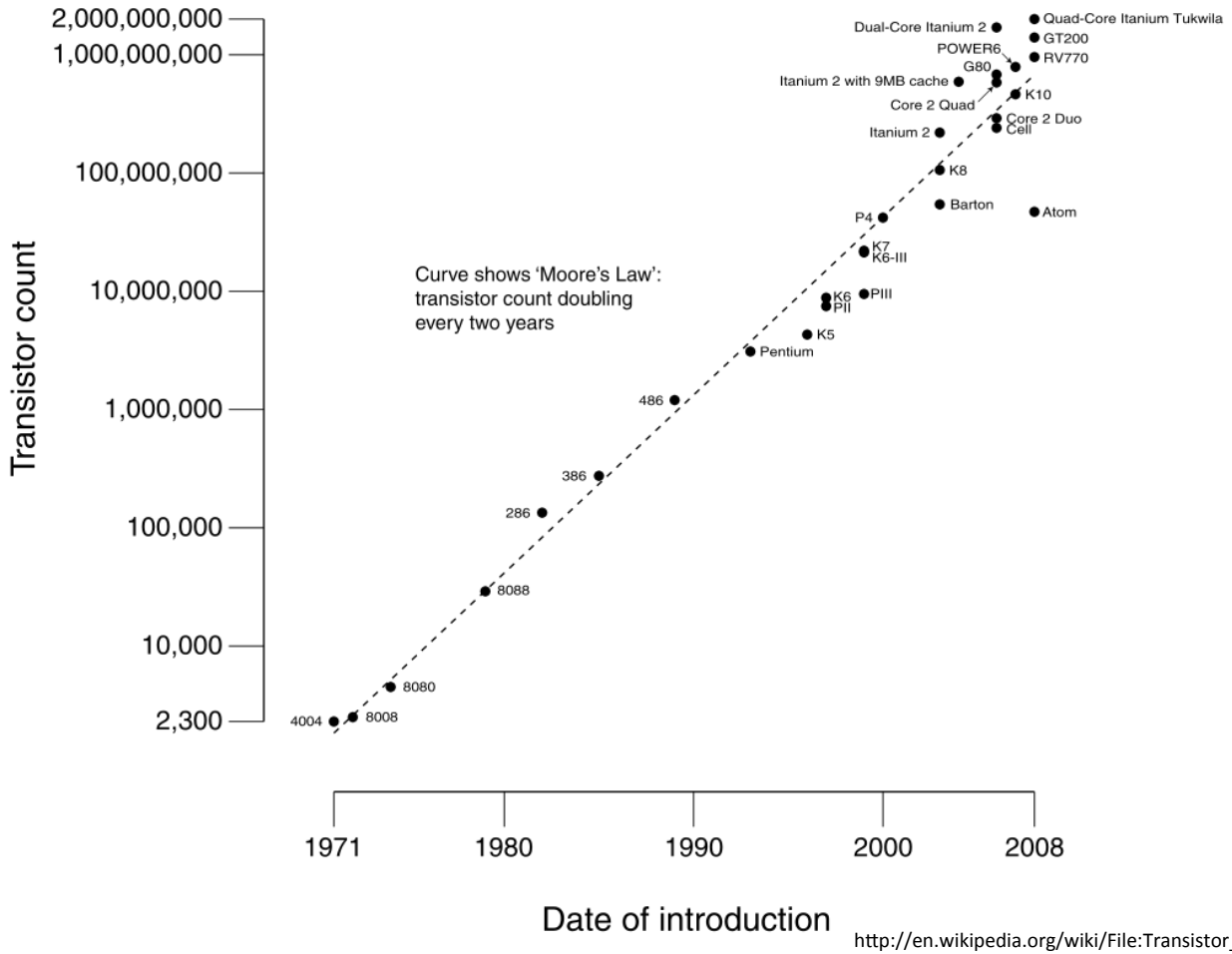
Presentation by: Andy Stone

April 27, 2010

Research Exam

Colorado State University

A Promising Growth Trend



http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2008.svg

A Promising Growth Trend

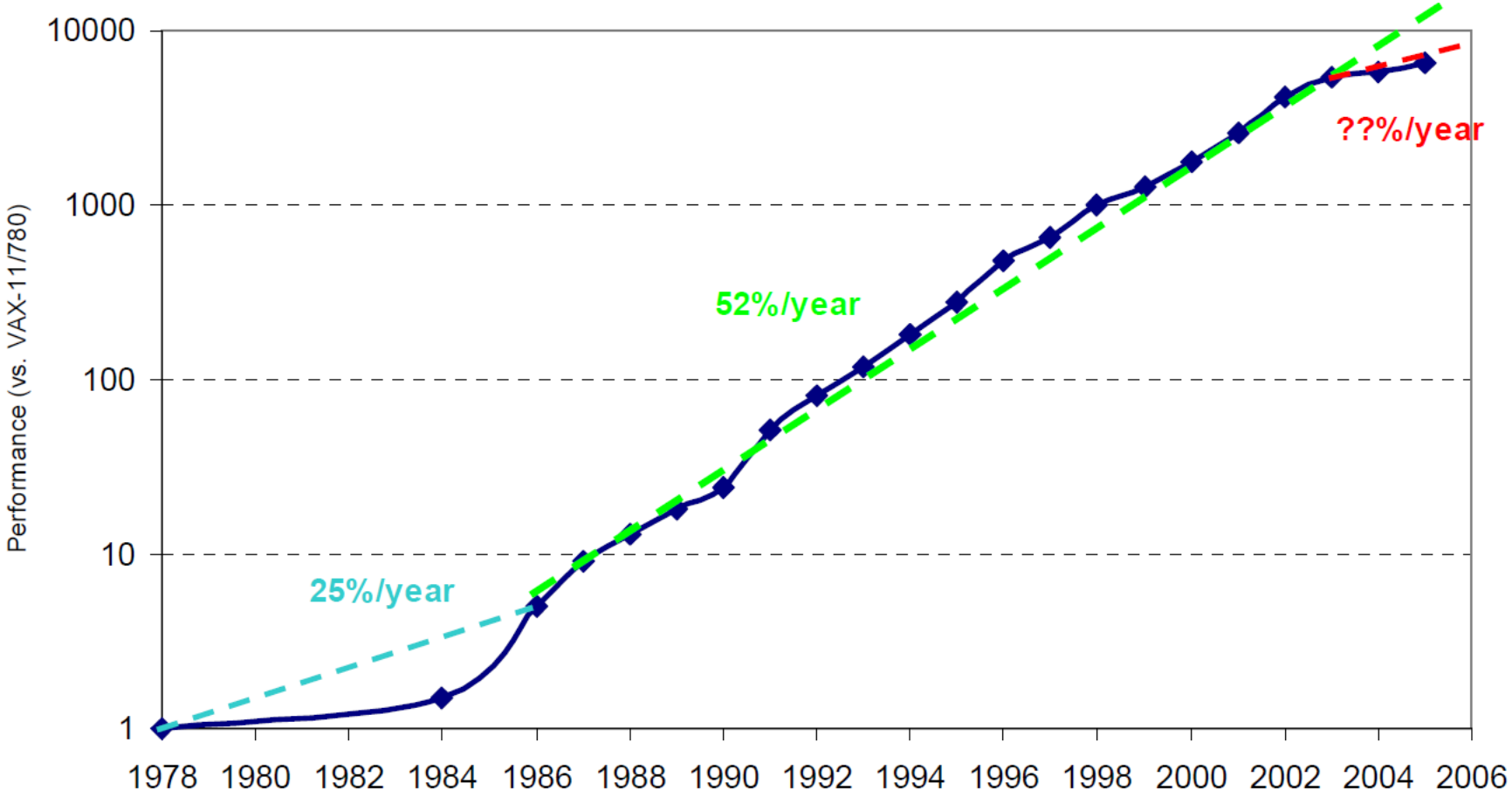
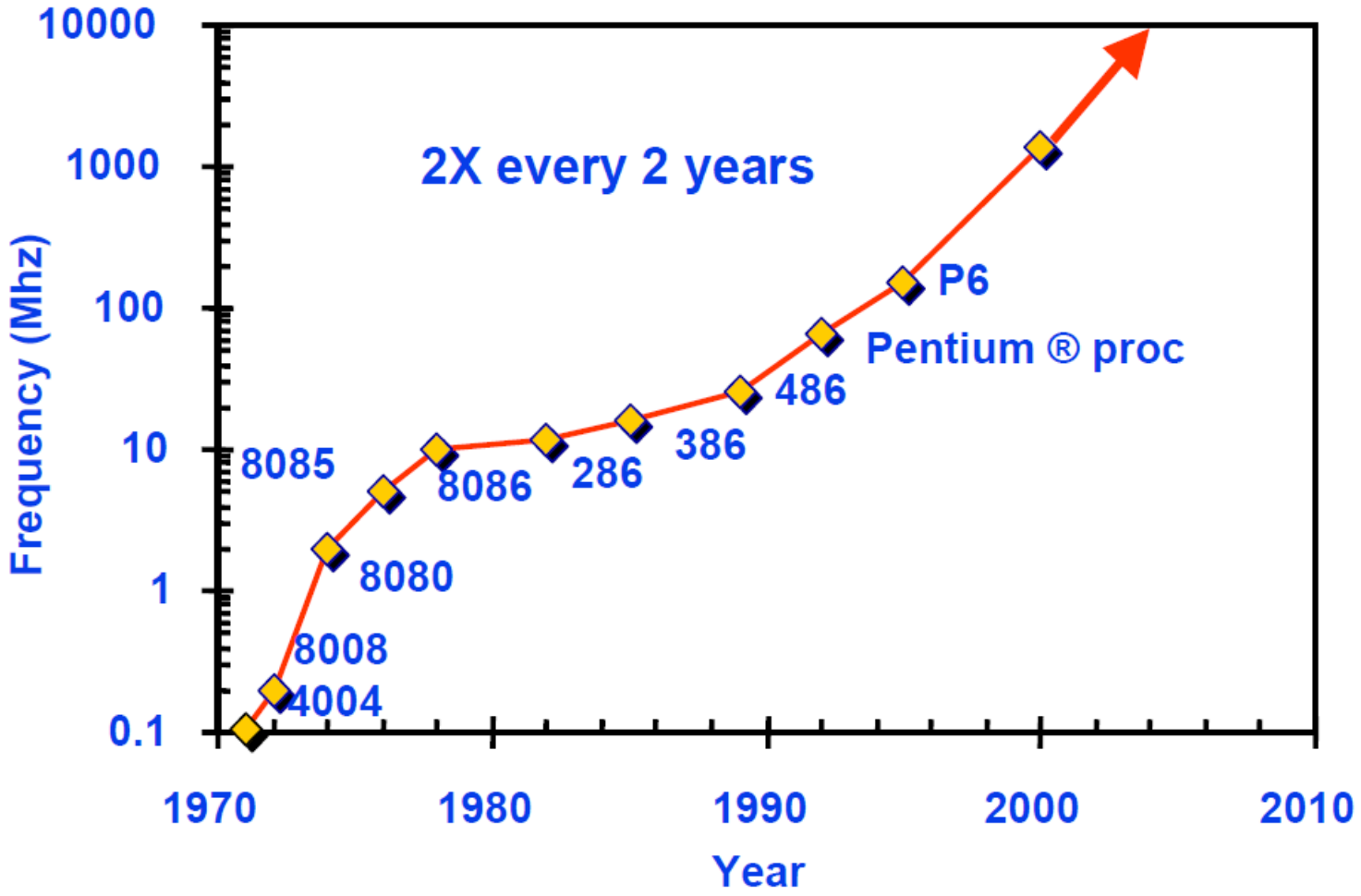


Image from Berkeley View White-Paper (The Landscape of Parallel Computing Research: A View from Berkeley) http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html
They take it from a book by Hennessy and Patterson: J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 4th eddition, Morgan Kauffman, San Francisco, 2007

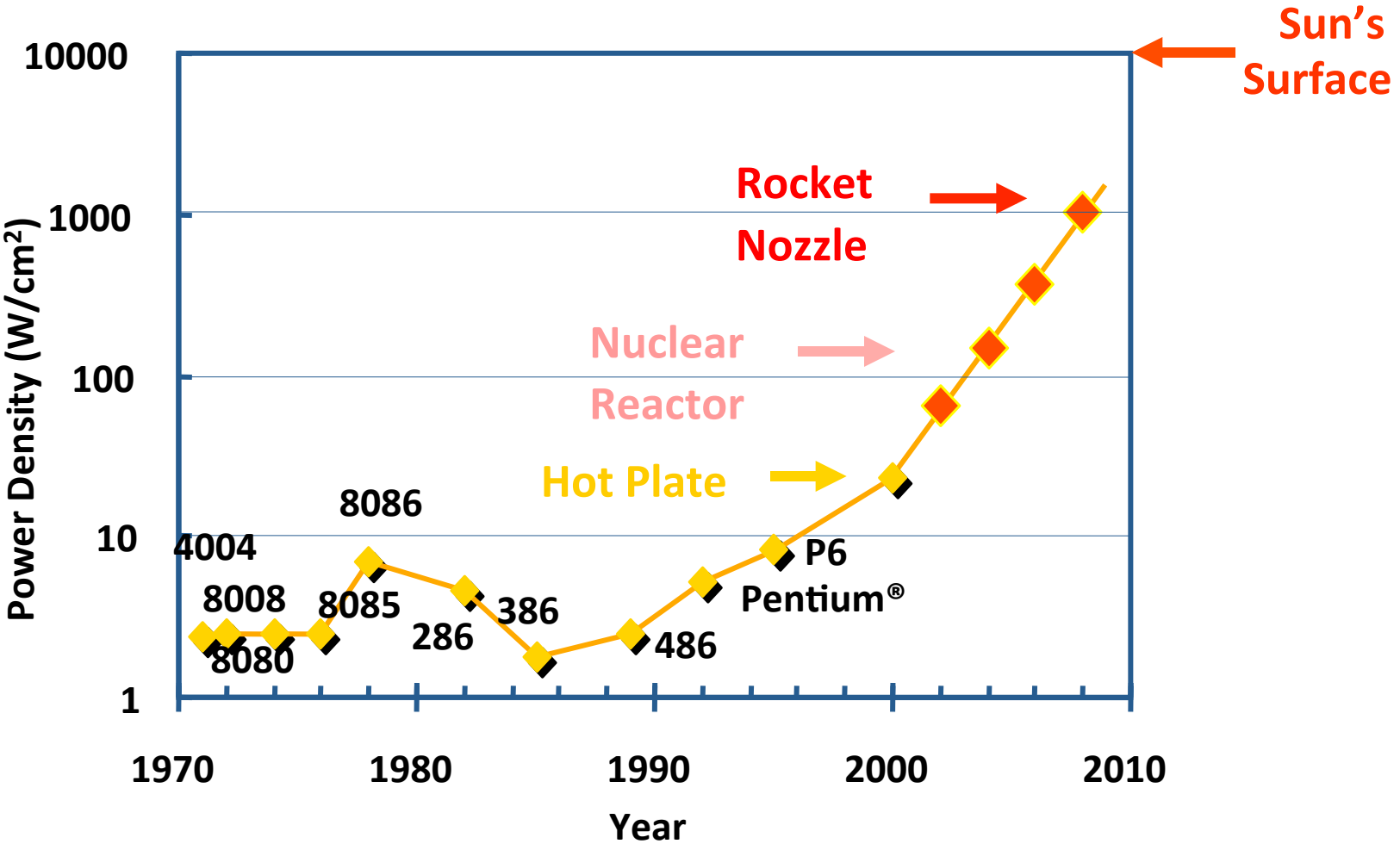
Another Growth Trend



http://www.nd.edu/~jbb/cse462_06/lectures/L03_Trends.pdf

Not so

A Promising Growth Trend

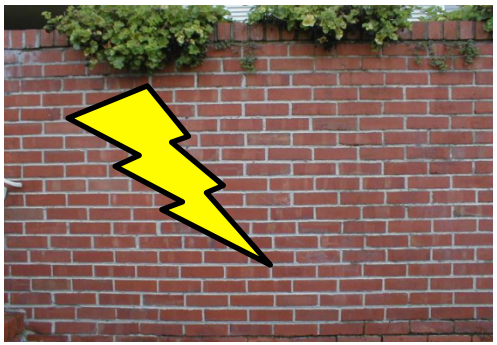


http://www.eng.auburn.edu/~agrawvd/D&TSEMINAR_SPR06/SLIDES/Agrawal_DTSem06.ppt

Source: Intel®

The power wall

The power wall is preventing further increases in processor operating frequency



How do I get more performance?

slow sauce

Freakin' fast

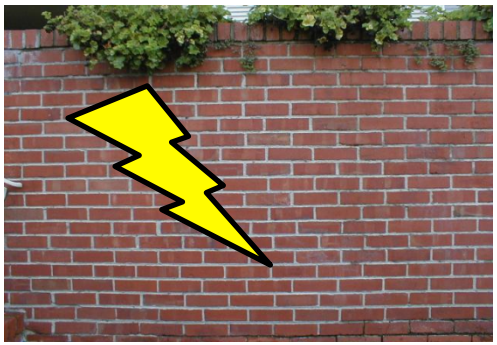
CPU Operating Frequency

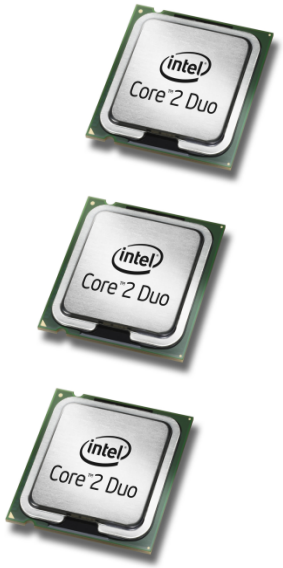
Images from:
http://www.math.toronto.edu/~drorbn/Gallery/Symmetry/Tilings/2S22/BrickWall.jpg
http://computer-reviews.net/files/Intel%20Pentium%204%20530%203.0GHz%20800MHz%20bus%20Socket%20775.jpg

How to get additional performance

Solution: Use more processing elements!

Don't worry CPU;
Just add some buddies:





Parallel Architectures:

Multicore!
SMP!
Clusters!
Cloud Computing!

slow sauce

Freakin' fast

CPU Operating Frequency

Images from:
http://www.math.toronto.edu/~drorbn/Gallery/Symmetry/Tilings/2S22/BrickWall.jpg
http://computer-reviews.net/files/Intel%20Pentium%204%20530%203.0GHz%20800MHz%20bus%20Socket%20775.jpg
http://cryptik.net/catalog/images/img_1614_intel-core2-duo.jpg

Surveying how Parallel Programming Models Address Computation Distribution

Insights on **what makes parallel programming difficult** and how to compare and judge programming models that aim to alleviate these difficulties

Presentation by: Andy Stone

April 27, 2010

Research Exam

Colorado State University

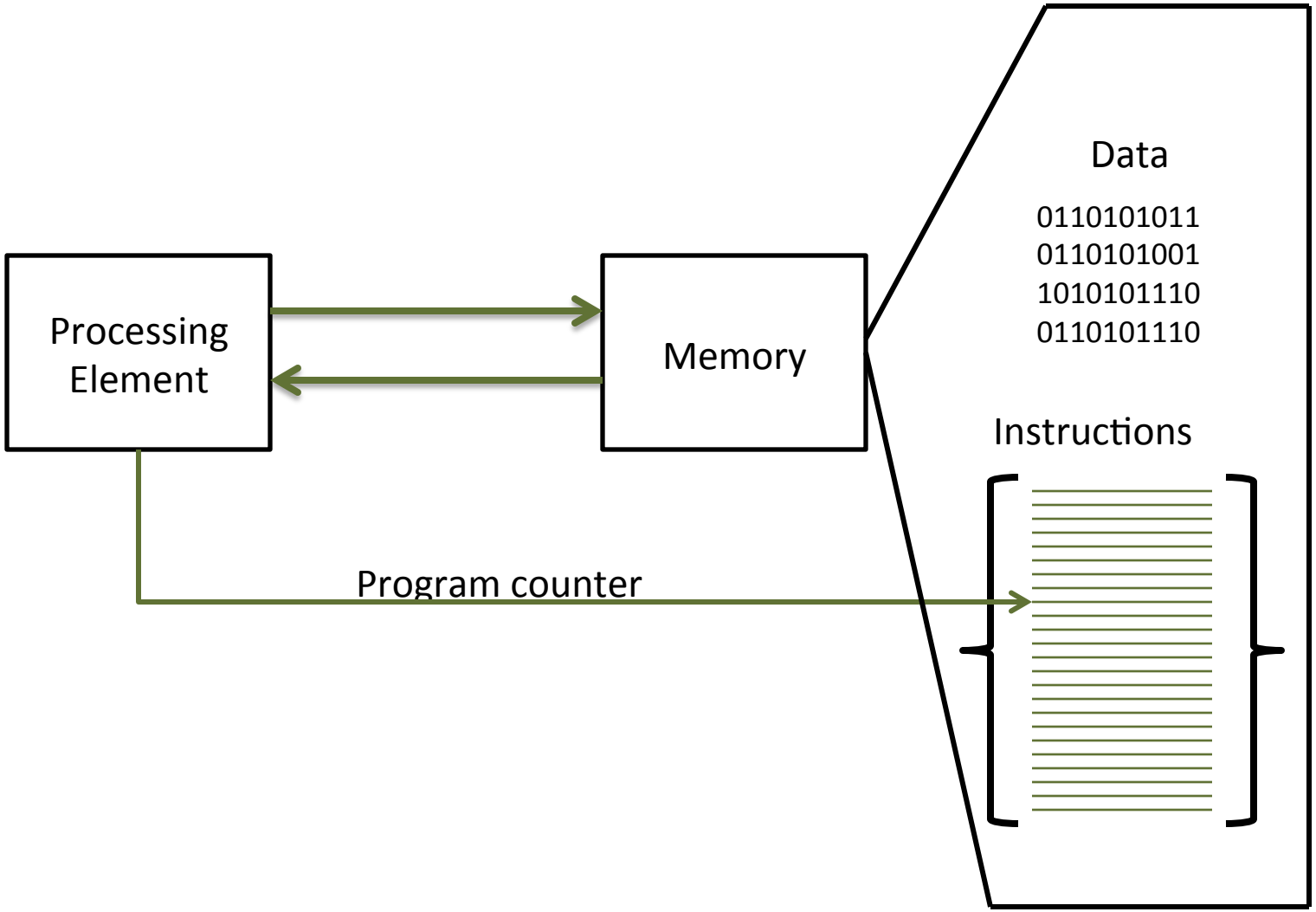
What is software going to look like?

Parallel software faces challenges unseen in serial software

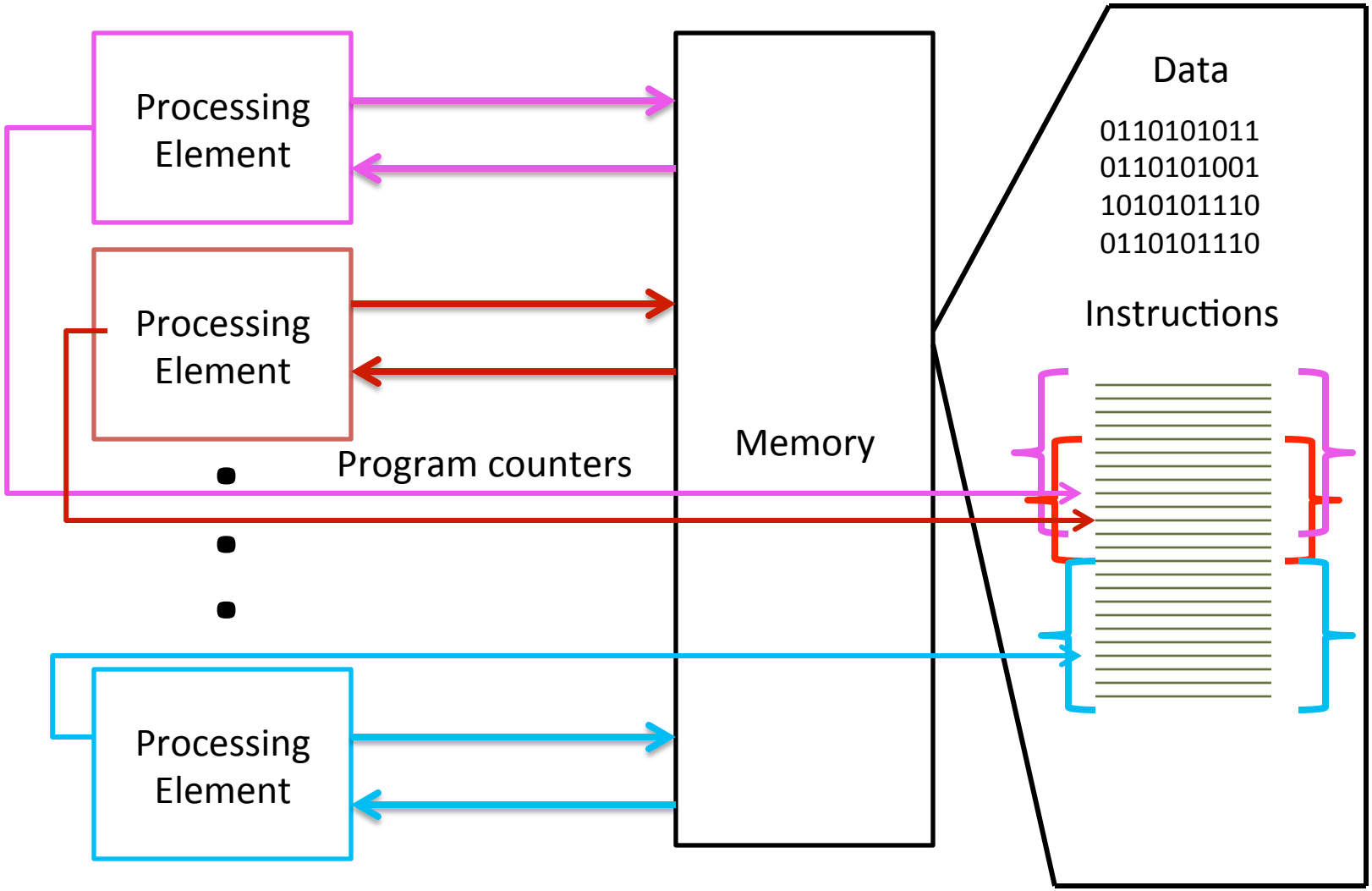
What are these challenges?

Think about how a parallel system differs from a serial system, and think of what changes are required to port software from a serial system to a parallel system

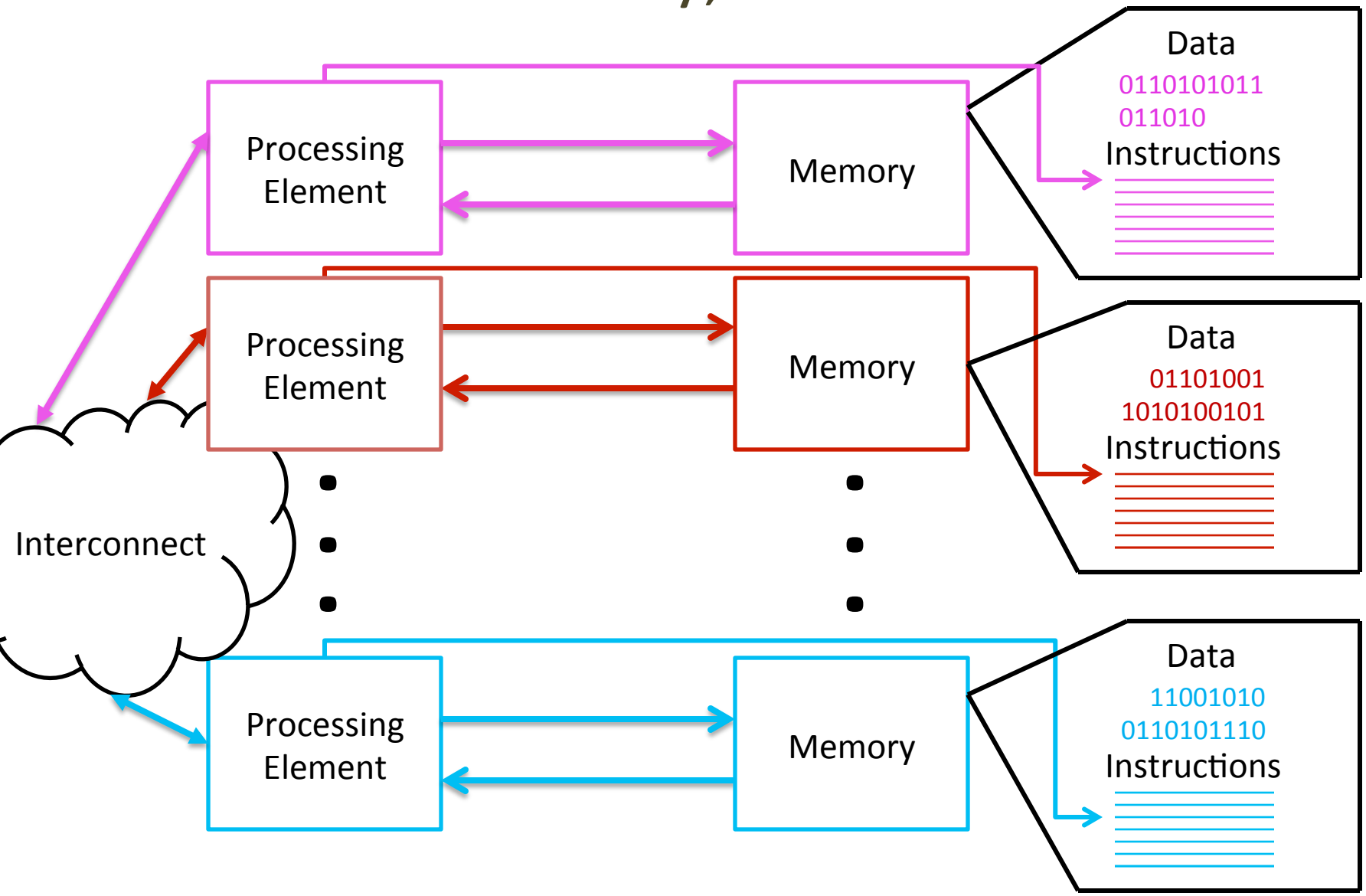
Serial Architecture



Shared Memory, Parallel Architecture



Distributed Memory, Parallel Architecture

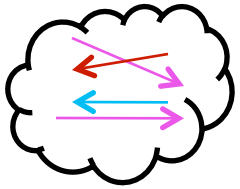


Three Factors

Distribution of Computation

0110101011
0110101001
1010101110
0110101110

Distribution of Data



Communication

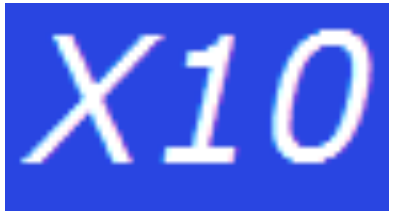
How to address these factors?

Programming models of course!

Cilk

Ct

CHAPEL



STAPL

Map-Reduce

CUDA

Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes parallel programming difficult and **how to compare and judge programming models** that aim to alleviate these difficulties

Presentation by: Andy Stone

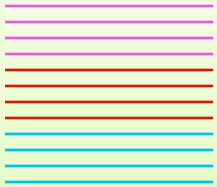
April 27, 2010

Research Exam

Colorado State University

Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes parallel programming difficult and **how to compare and judge the programming models** that aim to alleviate these difficulties



Distribution of Computation

Research Exam

Colorado State University

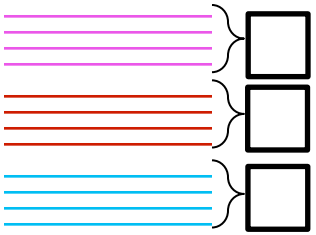
Two types of criteria:

Interpretive

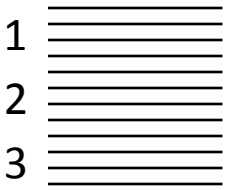
Evaluative

Image from: http://www.tulanelink.com/legal/balance2.gif

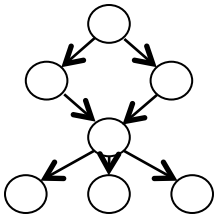
Four Interpretive Criteria



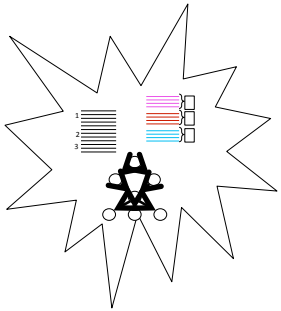
The specification of how computation is distributed across processing elements



The specification of how computation is distributed across time



The structure of computation



The realization of distributed computation

Evaluating programming models

Given the set of computation a model represents

How well can they run?

Where can they run?

What can they run?

Who can they run with?

What do they demand of the programmer?

Lots of Evaluative Criteria

How well can they run?

Performance

Fault-tolerance

Testability

Where can they run?

Portability

Lots and lots of Evaluative Criteria

What can they run?

Expressability

User-Control

Adaptability

Who can they run with?

Interoperability

Composability

Lots^3 of Evaluative Criteria

What do they demand of the programmer?

User-Responsibility

Clarity

Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes parallel programming difficult and how to compare and judge programming models that aim to alleviate these difficulties

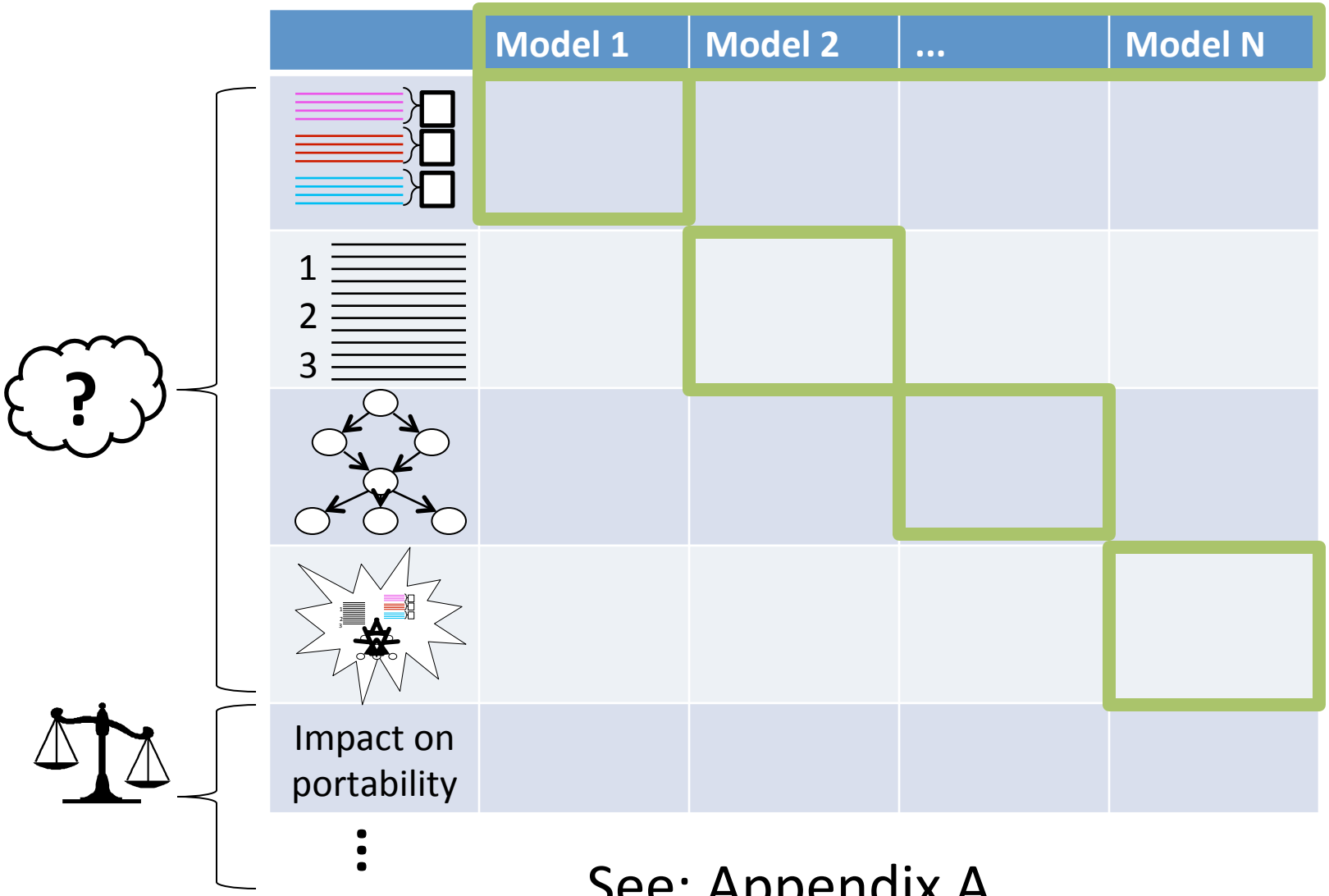
Presentation by: Andy Stone

April 27, 2010

Research Exam

Colorado State University

Structure of the Survey



See: Appendix A

The programming models

Iterators In Chapel

Sequoia

Granules

Id-Nouveau

Tang and Xue's Model for Tiled Code Generation

The programming models

Iterators In Chapel

Sequoia

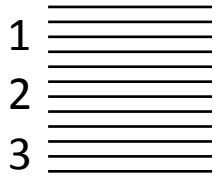
Granules

Id-Nouveau

Tang and Xue's Model for Tiled Code Generation

Separating iteration logic from loop-body

M. Joyner, B. L. Chamberlain, and S. J. Deitz. Iterators in chapel. In 11th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS), April 2006.

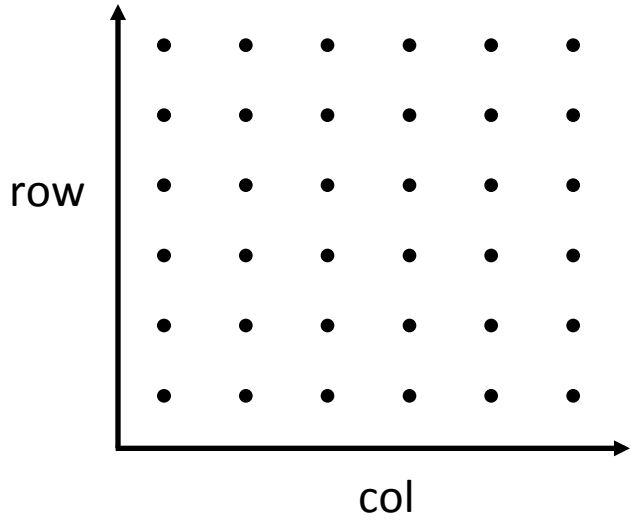


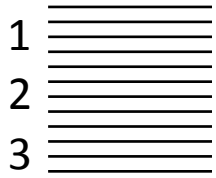
Chapel iterators

Specification of comp. dist across time

```
for (row,col) in a 6x6 iteration space {
        writeln("At index: ", row, " ", col);
}
```

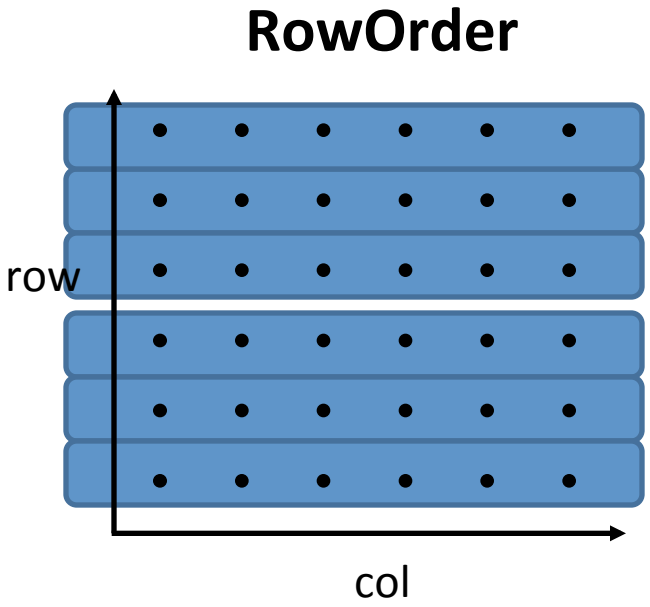
How to step through this iteration space?

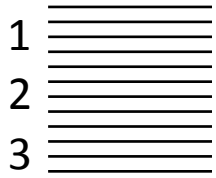




Chapel iterators

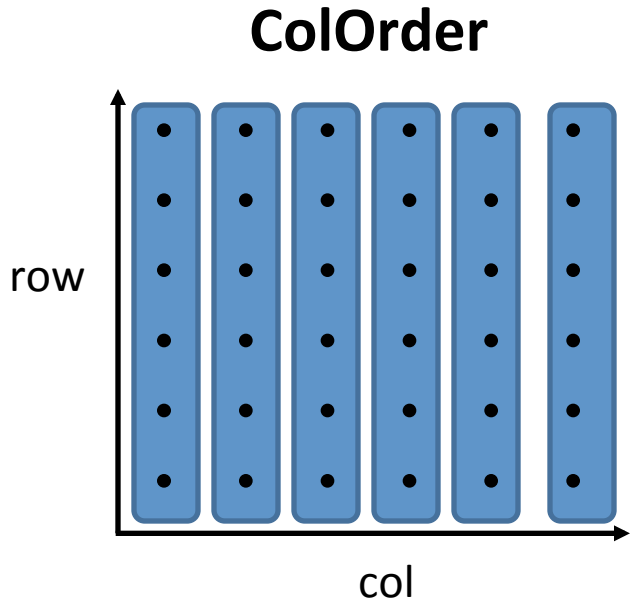
Specification of comp. dist across time

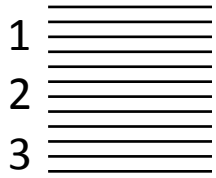




Chapel iterators

Specification of comp. dist across time





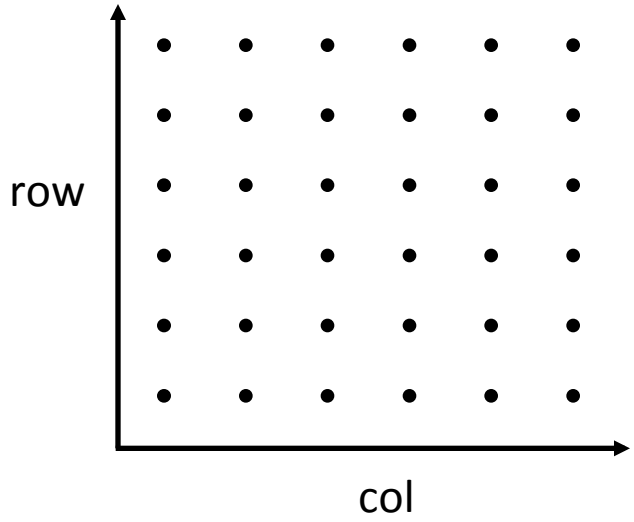
Chapel iterators

Specification of comp. dist across time

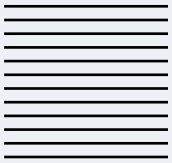
```
for (row,col) in a rowOrder(6,6) {
       writeln("At index: ", row, " ", col);
}

def rowOrder(nRows, nCols) {
    for row in 1..nRows
        forall col in 1..nCols
            yield (row, col);
}

def colOrder(nRows, nCols) {
    for col in 1..nCols
        forall row in 1..nRows
            yield (row, col);
}
```



Chapel, Evaluated

	Chapel
	
1 2 3 	Iteration logic specified in an iterator (written like a function)
	
	
Portability	Separation of iteration logic enables new logic to easily be added and changed; if iteration logic is machine dependent this separation aids in portability.
Expressability	Iterators are written in same language as algorithms
Composability	Iterators can be embedded in other iterators
Clarity	Separation of concerns improves clarity

The programming models

Iterators In Chapel

Sequoia

Granules

Id-Nouveau

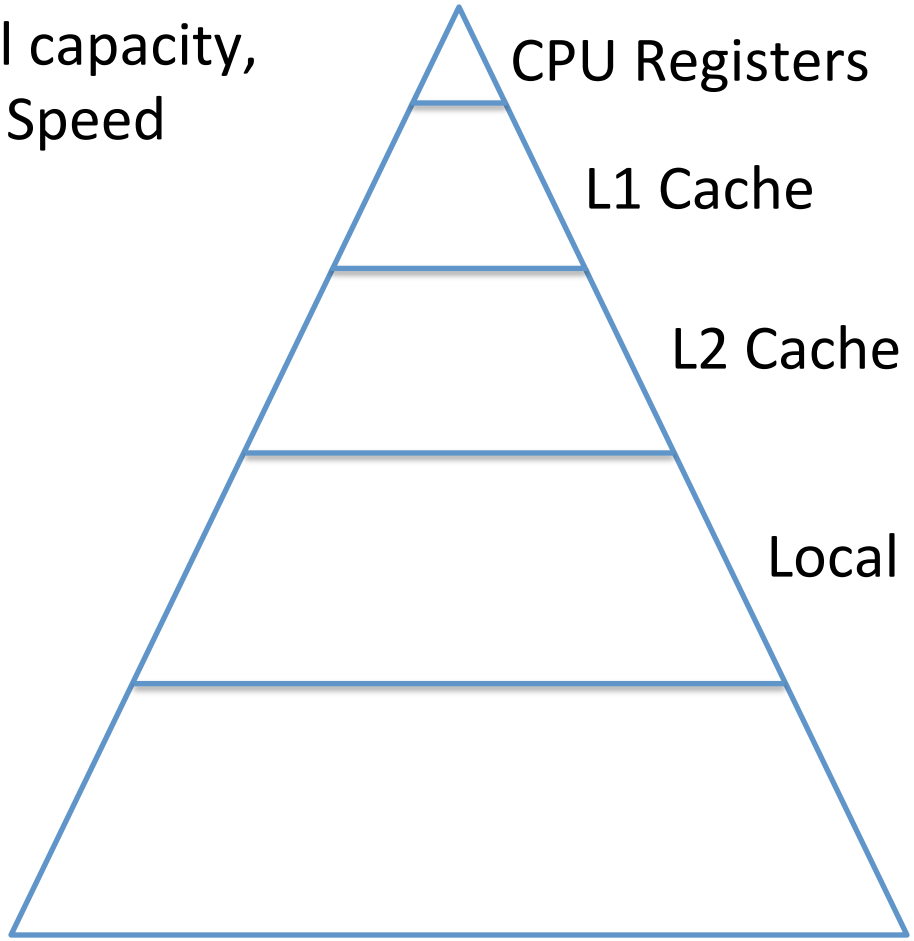
Tang and Xue's Model for Tiled Code Generation

Addressing the memory hierarchy

K. Fatahalian, T. J. Knight, M. Houston, M. Erez, D. R. Horn, L. Leem, J. Y. Park, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: Programming the memory hierarchy. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 2006.

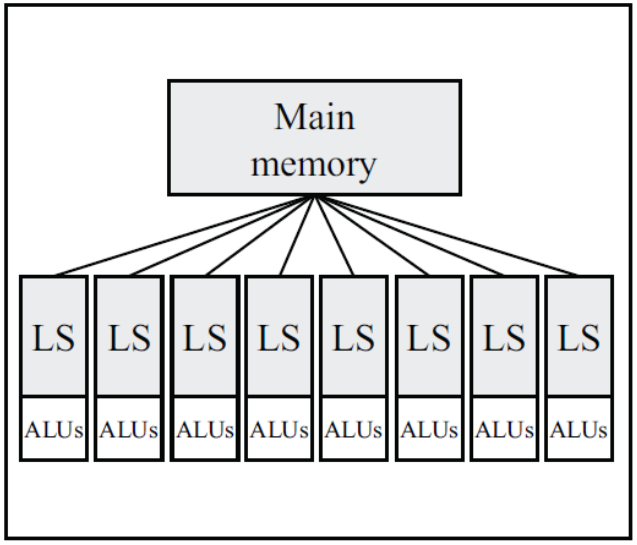
Sequoia

The Memory Hierarchy

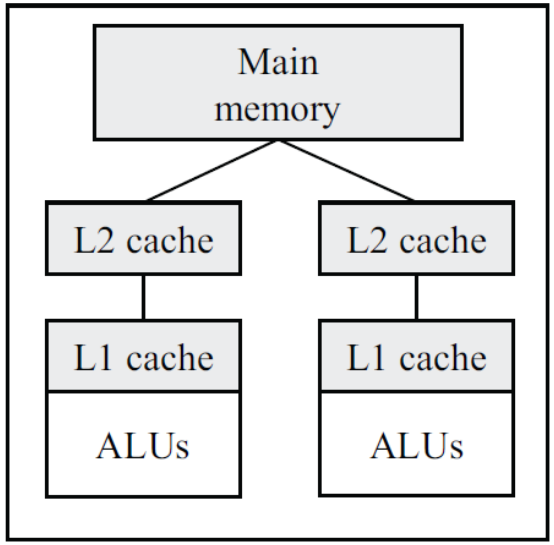


Sequoia

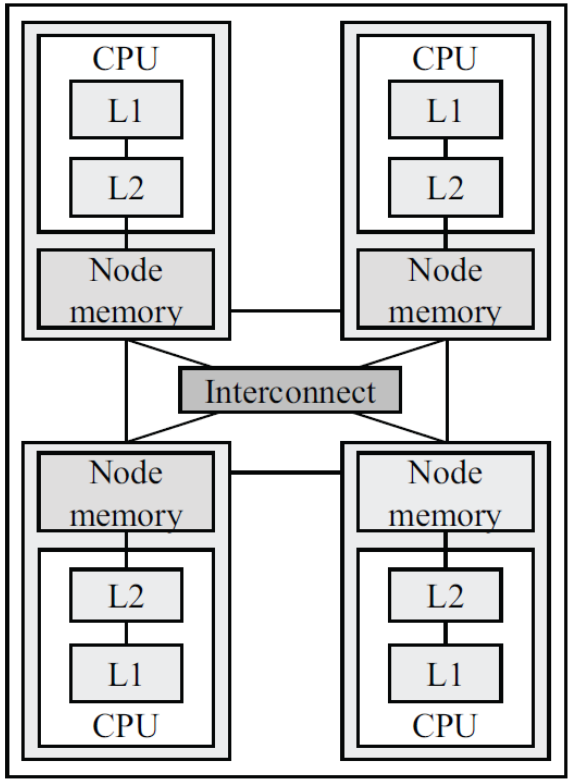
Arrangements of Memory Hierarchy



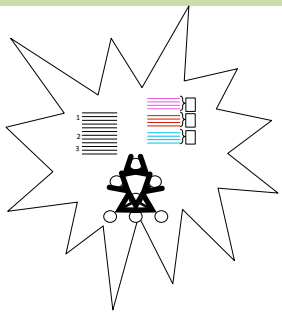
CELL processor



Dual-CPU workstation

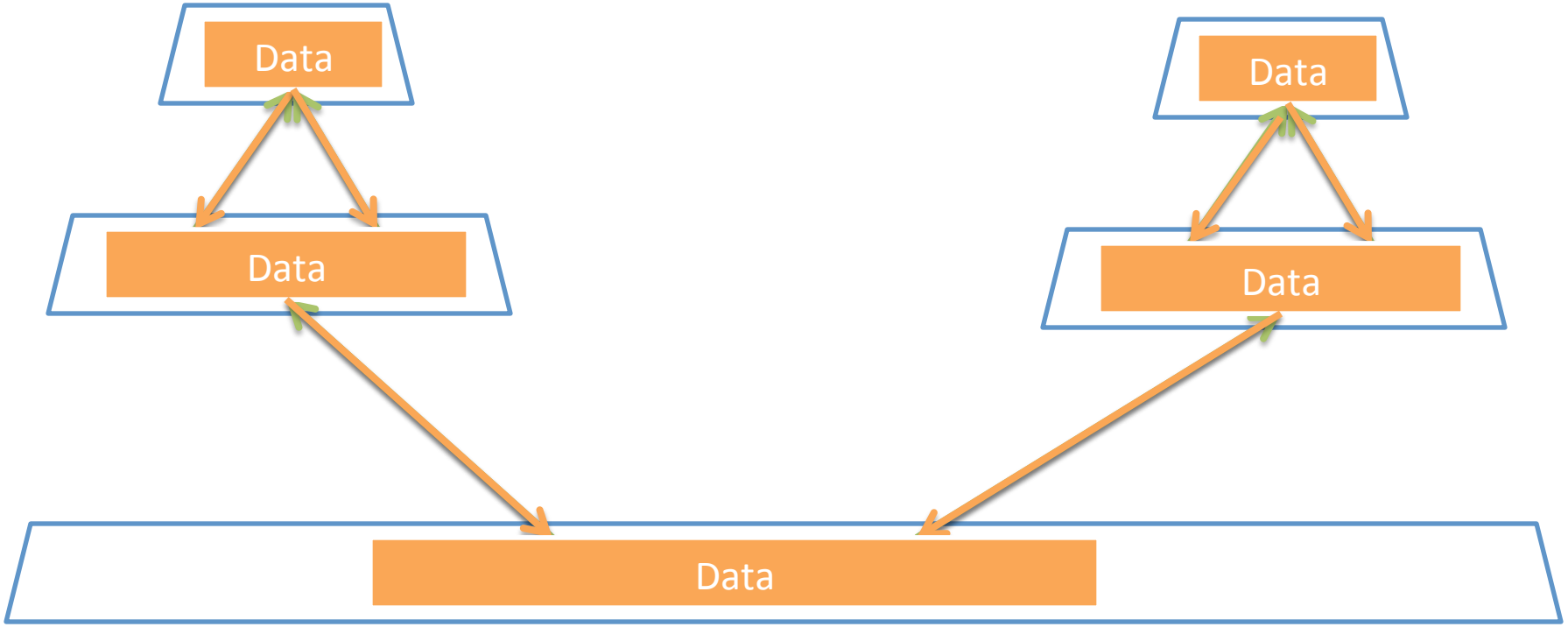


Cluster of uniprocessor PCs

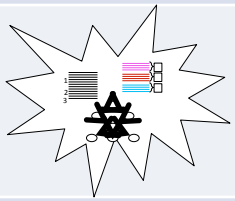


Sequoia

Realization of Distributed computation



Sequoia, Evaluated

	Sequoia
	
	
	
	Recursively decompose data to fit on successive layers of hierarchy.
Portability	Memory hierarchies are machine dependent. Application can be ported to operate with different memory hierarchies by modifying mapping specification files (that describe this hierarchy)
Expressability	Limited to decomposition of arrays.
Testability	No clear way to tell if data is actually moving to appropriate levels of memory hierarchy.

The programming models

Iterators In Chapel

Sequoia

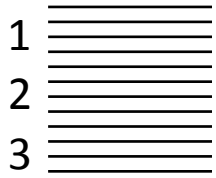
Granules

Id-Nouveau

Tang and Xue's Model for Tiled Code Generation

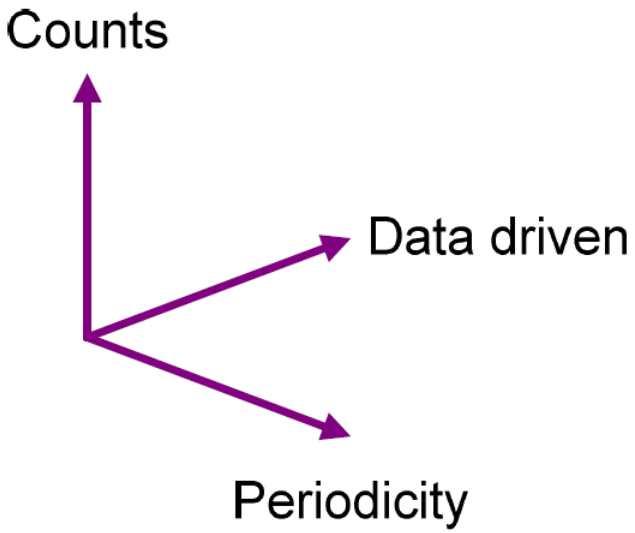
Have runtime allocate tasks to resources in the cloud.

S. Pallickara, J. Ekanayake, and G. Fox. An overview of the granules runtime for cloud computing. In Proceedings of the IEEE International Conference on e-Science, December 2008.



Granules

Specification of comp. dist across time



Counts:

How many times should a task execute

Data driven

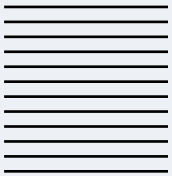
Execute on data-availability

Periodicity

Execute every n milliseconds

Can also express termination condition

Granules, Evaluated

	Sequoia
	
1 2 3 	Specified via scheduling strategies
	
	
Portability	Runtime system moves computation to computing resources, updates in runtime system to operate with new types of resources
Expressability	Expressability improved by enabling a combination and conditionality of counts/data-driven/periodicity parameters

The programming models

Iterators In Chapel

Sequoia

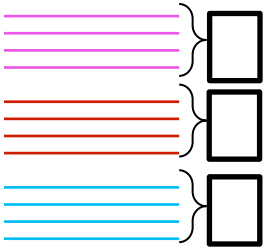
Granules

Id-Nouveau

Tang and Xue's Model for Tiled Code Generation

Determine computation distribution from an explicit decomposition of data

A. Rogers and K. Pingali. Process decomposition through locality of reference. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 69{80, New York, NY, USA, 1989. ACM.



Id-Nouveau

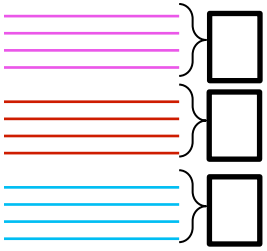
Specification of comp. dist. across procs.

Computation distribution follows from data distribution

Compiler automatically determines how to distribute computation

The data structure of Id-Nouveau is the I-Structure

Has single assignment semantics



Id-Nouveau

Specification of comp. dist. across procs.

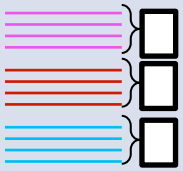
Array Mapping

Map: Maps indices to processing element

Local: Maps index to local element

Alloc: Allocates the array

Id-Nouveau, Evaluated

	Sequoia
	Explicitly distribute data, computation distribution follows from the data-distribution
	
	
	
Expressability	Limited to operating on arrays with single-assignment semantics
Testability	Improved since code maintains serial semantics
Clarity	Code appears serial

The programming models

Iterators In Chapel

Sequoia

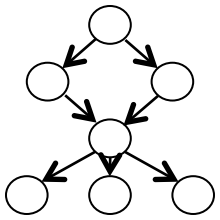
Granules

Id-Nouveau

Tang and Xue's Model for Tiled Code Generation

Define a geometric structure to computation, distribute computation in that structure

P. Tang and J. Xue. Generating efficient tiled code for distributed memory machines. Parallel Computing, 26(11):1369 { 1410, 2000.



Tang and Xue's Model for Tiled Code

Structure of Computation

$$\text{for}(i_1 = L_1;\ i_1 \leq U_1;\ i_1\text{++})$$
$$\ldots$$
$$\text{for}(i_n = L_n;\ i_n \leq U_n;\ i_n\text{++})$$
$$A(f(\vec{i})) = F(A(f(\vec{i} - \vec{d_1})),\ \ldots,\ A(f(\vec{i} - \vec{d_r})))$$

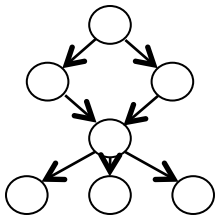
Where:

A is an array.

F is a (side-effect free) function.

i_1 through i_n are loop index variables.

L_1 through L_n and U_1 through U_n are affine expressions of the loop index variables above them.



Tang and Xue's Model for Tiled Code

Structure of Computation

$$\text{for}(i_1 = L_1;\ i_1 \leq U_1;\ i_1\text{++})$$
$$\ldots$$
$$\text{for}(i_n = L_n;\ i_n \leq U_n;\ i_n\text{++})$$
$$A(f(\vec{i})) = F(A(f(\vec{i} - \vec{d_1})),\ \ldots,\ A(f(\vec{i} - \vec{d_r})))$$

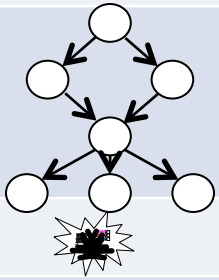
Where:

f is an affine expression of loop variables.

$\vec{i}$ is a vector of the loop variables i_1 through i_n.

$\vec{d_1}$ through $\vec{d_r}$ are constant dependence vectors. It is assumed that for all k 1..r $\vec{d_k} \in \mathbb{Z}^n$. It is also assumed that $r \geq n$.

Tang and Xue's Model, Evaluated

	Sequoia
	
	
	See previous two slides
Portability	Could be improved with a more espressible model (multi-level tiling, other mapping strategies, etc.)
Expressability	Can only handle singe assignments and uniform loop-carry dependencies

Surveying how Parallel Programming Models Address Computation Distribution

Insights on what makes parallel programming difficult and how to compare and judge programming models that aim to alleviate these difficulties

Presentation by: Andy Stone

April 27, 2010

Research Exam

Colorado State University

Conclusions

Examining programming models in terms of **interpretive** and **evaluative** criteria aids in **comparing** and **judging** them

In some systems computation distribution follows from data distribution

- This is true in Id-Nouveau and Sequoia not in Tang and Xue's model

Computation distribution is not an isolated issue

- I had to talk about data distribution in order to explain how to specify comp. distribution in Id-Nouveau

Conclusions

Separating machine-specific concerns from algorithm improves program portability

Chapel and Sequoia illustrate this point nicely.

Adaptability is influenced by how closely a model's structure of computation matches the structure of the program that is being adapted.

- Loops are easy to port to Chapel
- Porting to Id-Nouveau will be hard if your program isn't written with single-assignment semantics in mind
- If porting to fit into Tang & Xue's model you better have uniform dependencies.

Future Work

New and improved models:
Identify limitations in model according to the identified evaluative criteria, change the model to overcome these limitations or come up with something new that will

Finding better ways of comparing and judging models:
Survey how models address other two factors
Examine patterns of computation and how models help them

Acknowledgements

Committee members

Michelle Strout
Daniel Massey
Shrideep Pallickara

Editorial assistance

Alan Lamielle
Jon Roelofs

Other research exams

Chris Krieger

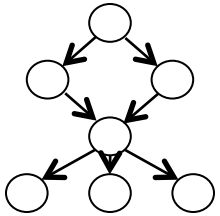
Chinese surname pronunciation help

Simon Sui

Practice presentation audience

Programming models for HPC reading-group

BONUS SLIDES



Sequoia
Structure of Computation

Language includes:

tunable parameters

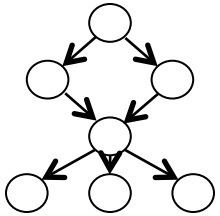
set to values specific to level of hierarchy computation is at

blocking primitives

decompose data (to fit on successive layers of hierarchy).

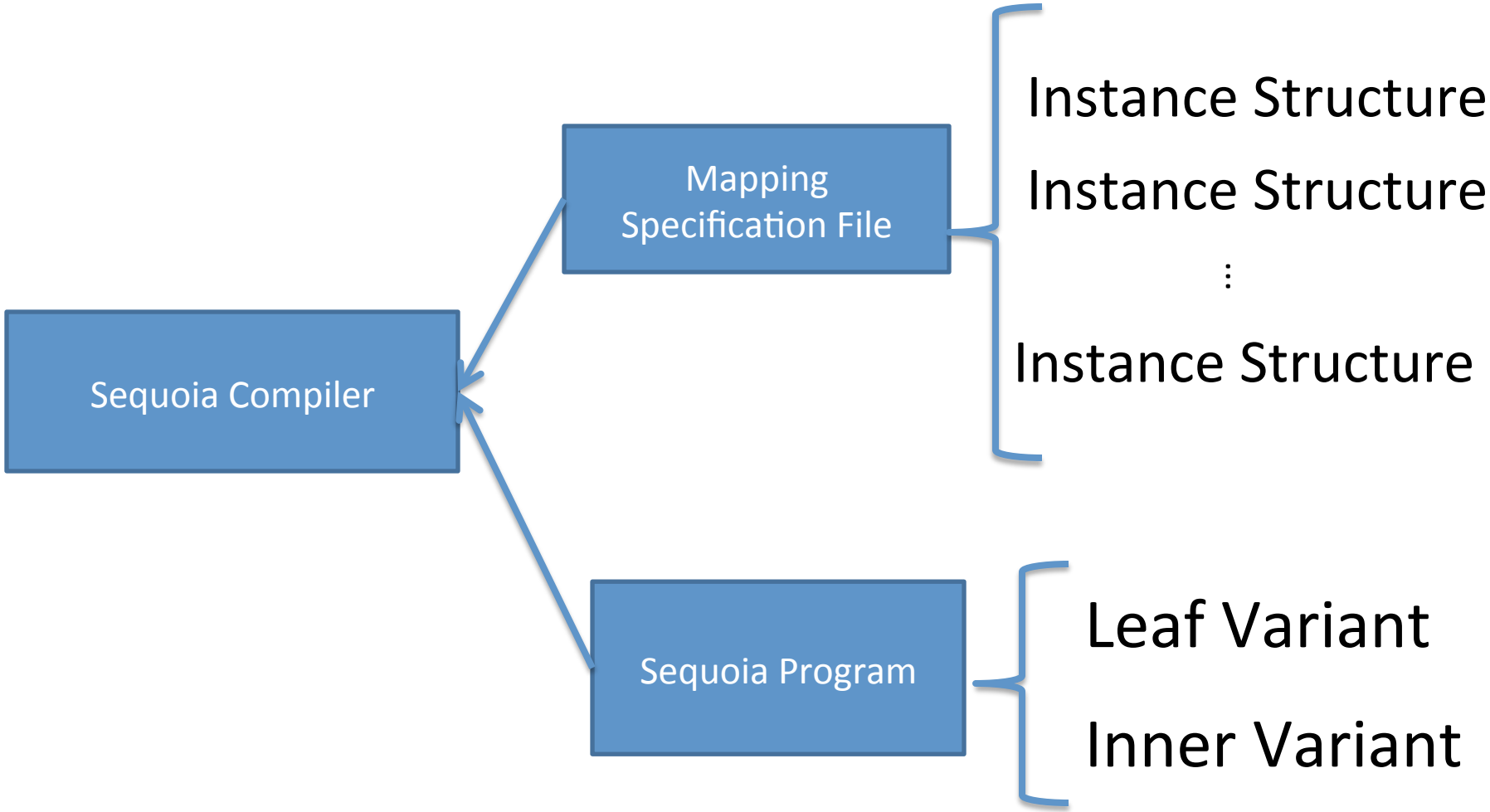
mapping primitives

spawn subtasks to work on decomposed data

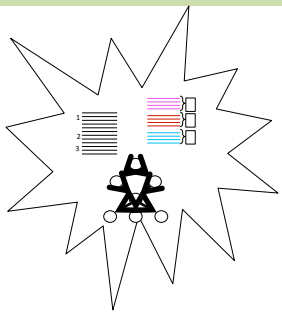


Sequoia

Structure of Computation

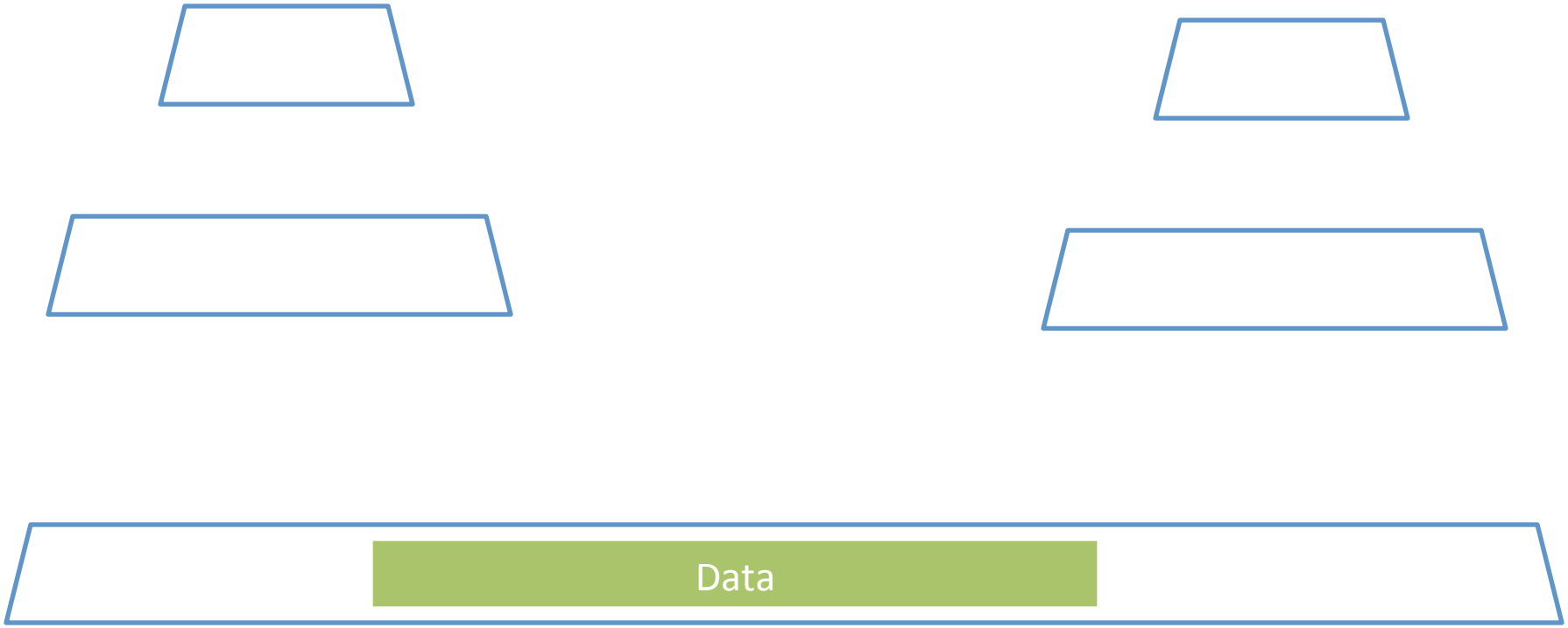


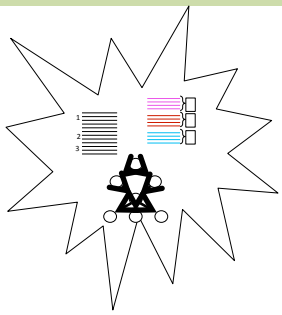
ANIMATION SLIDES



Sequoia

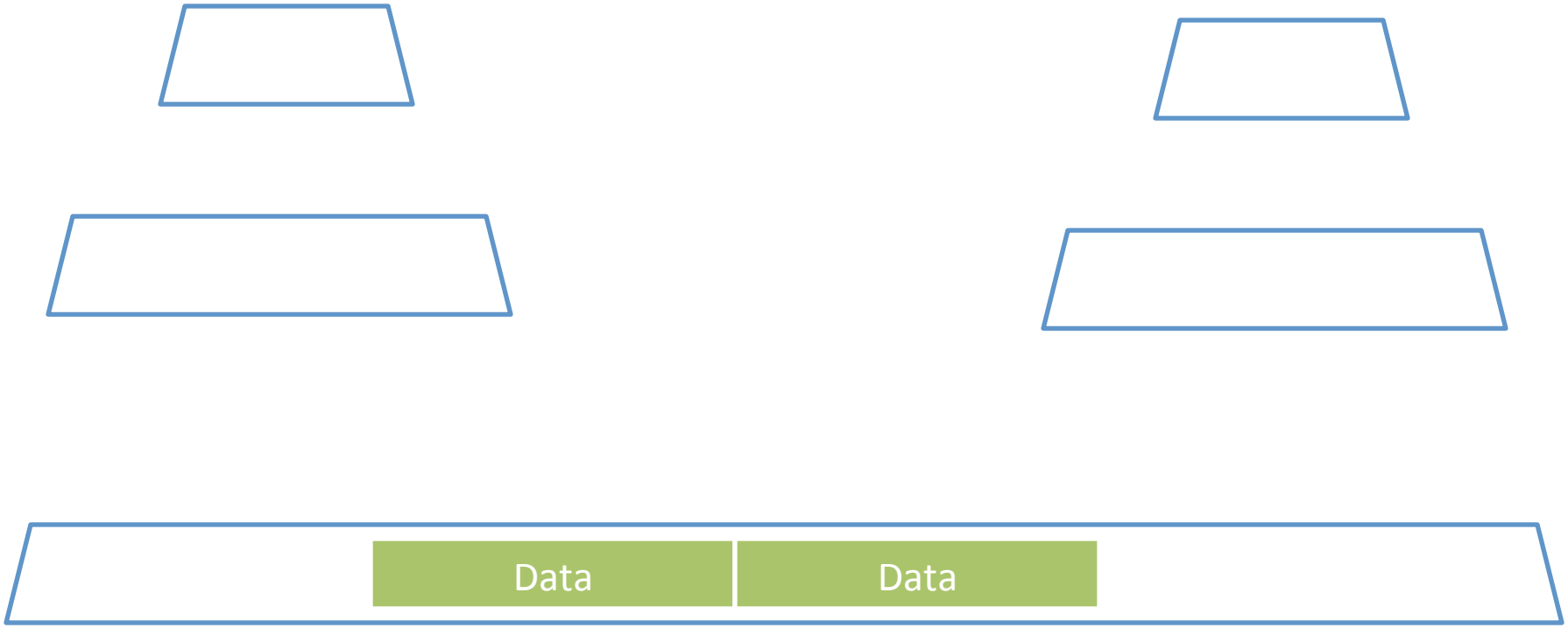
Realization of Distributed computation

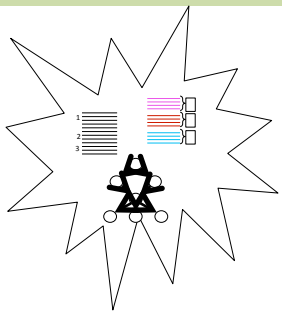




Sequoia

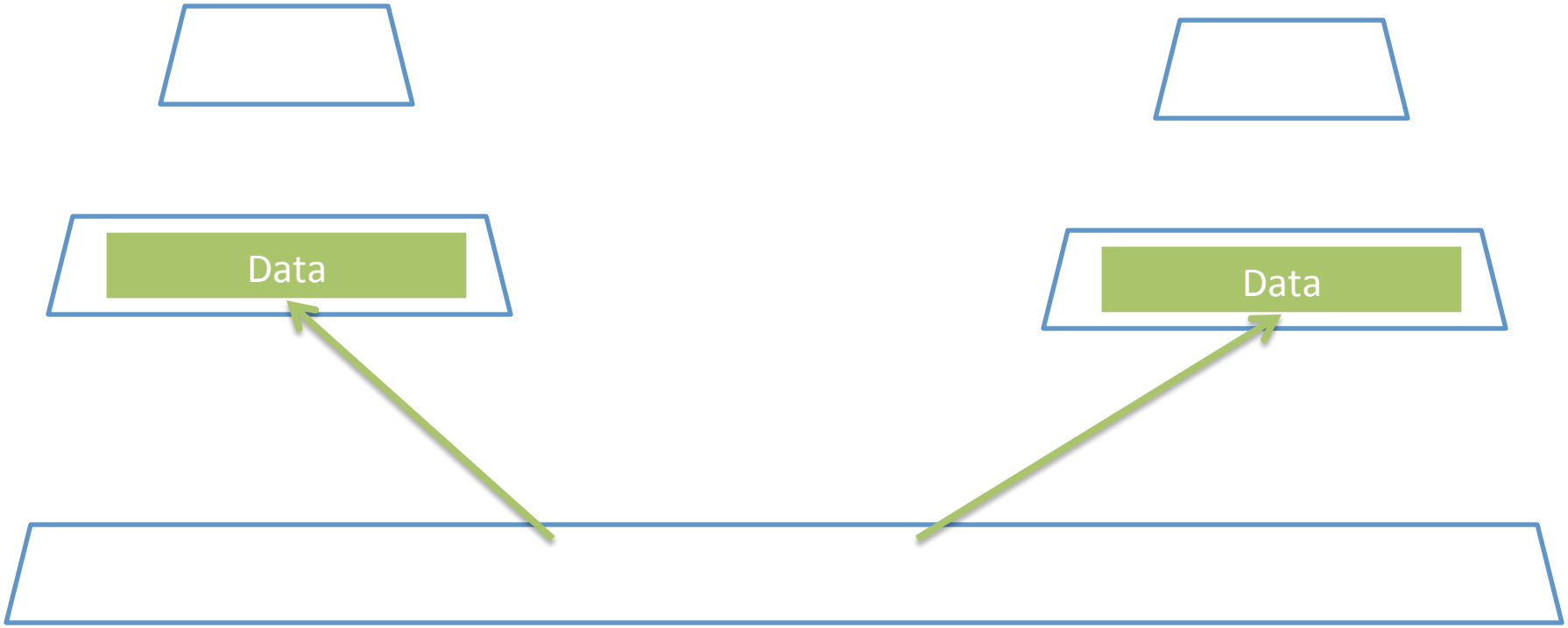
Realization of Distributed computation

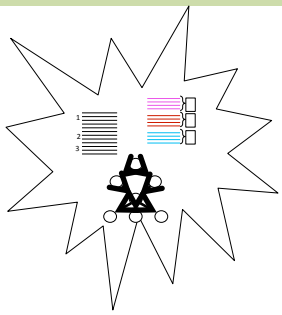




Sequoia

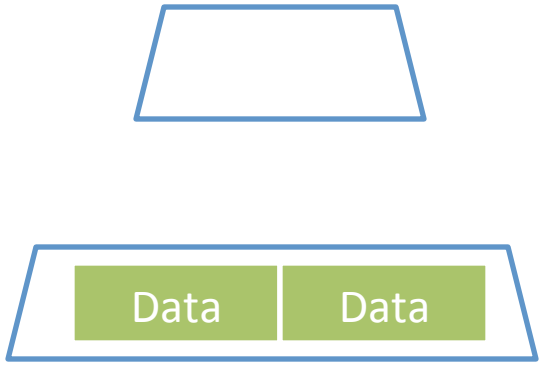
Realization of Distributed computation

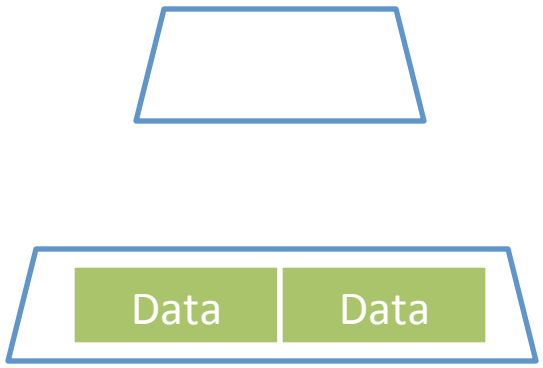


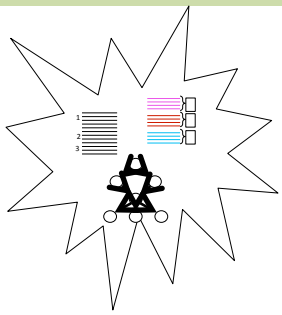


Sequoia

Realization of Distributed computation

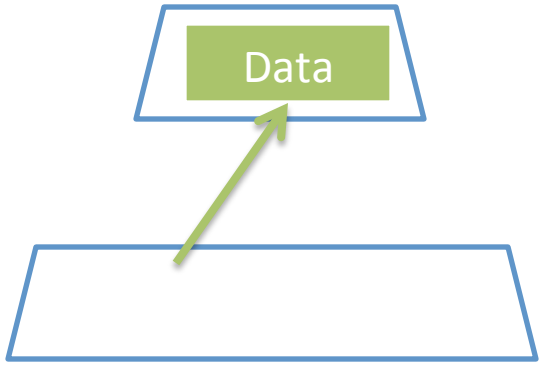


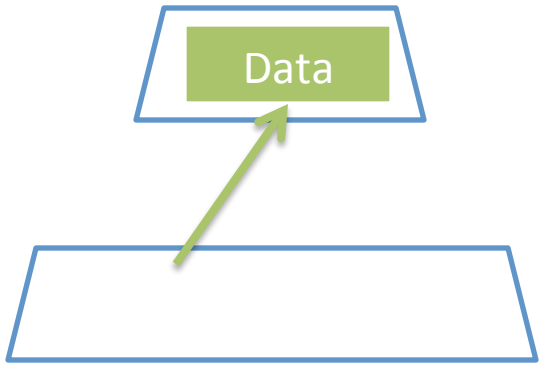


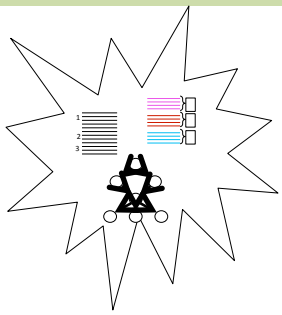


Sequoia

Realization of Distributed computation

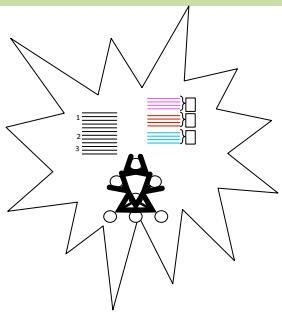






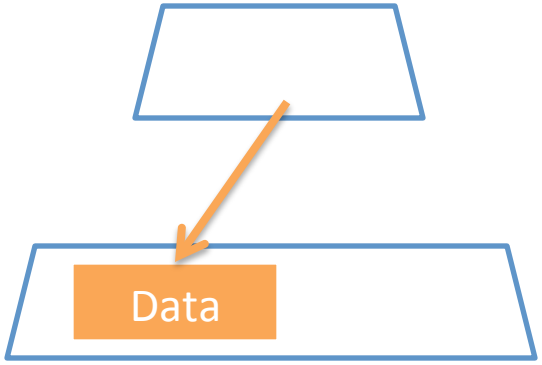
Sequoia

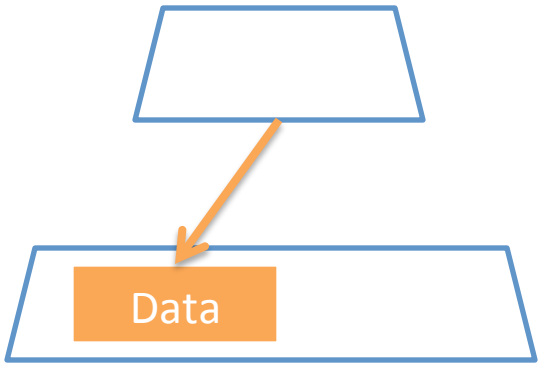
Realization of Distributed computation

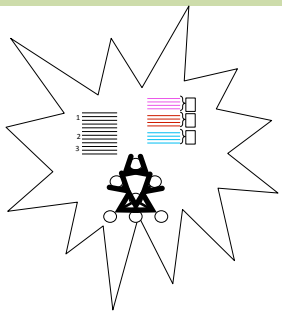


Sequoia

Realization of Distributed computation

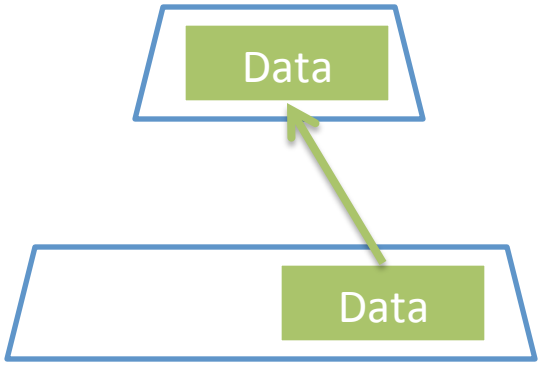


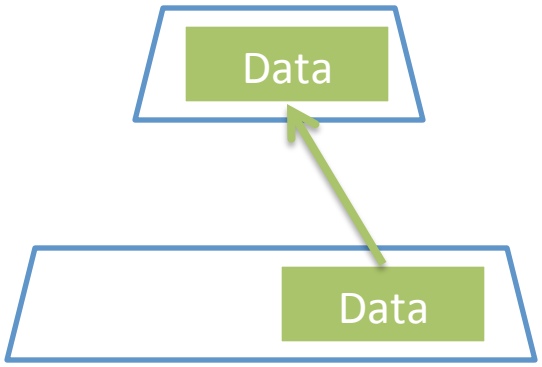


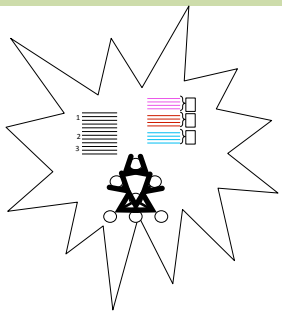


Sequoia

Realization of Distributed computation

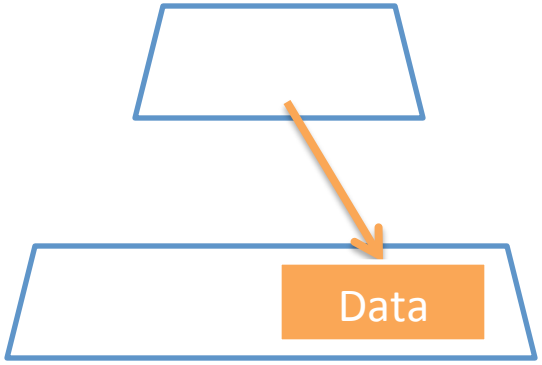


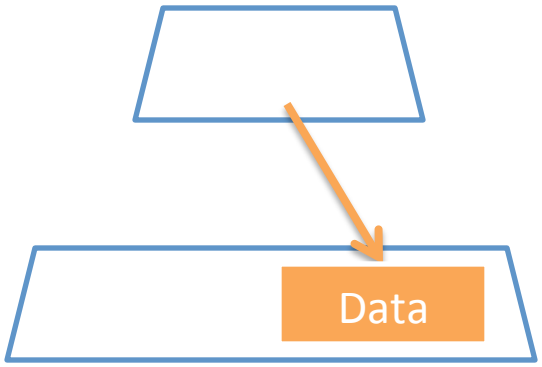


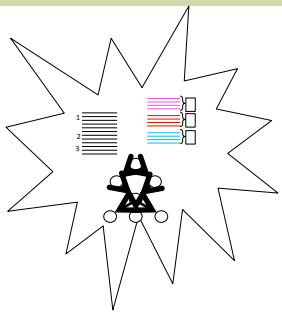


Sequoia

Realization of Distributed computation

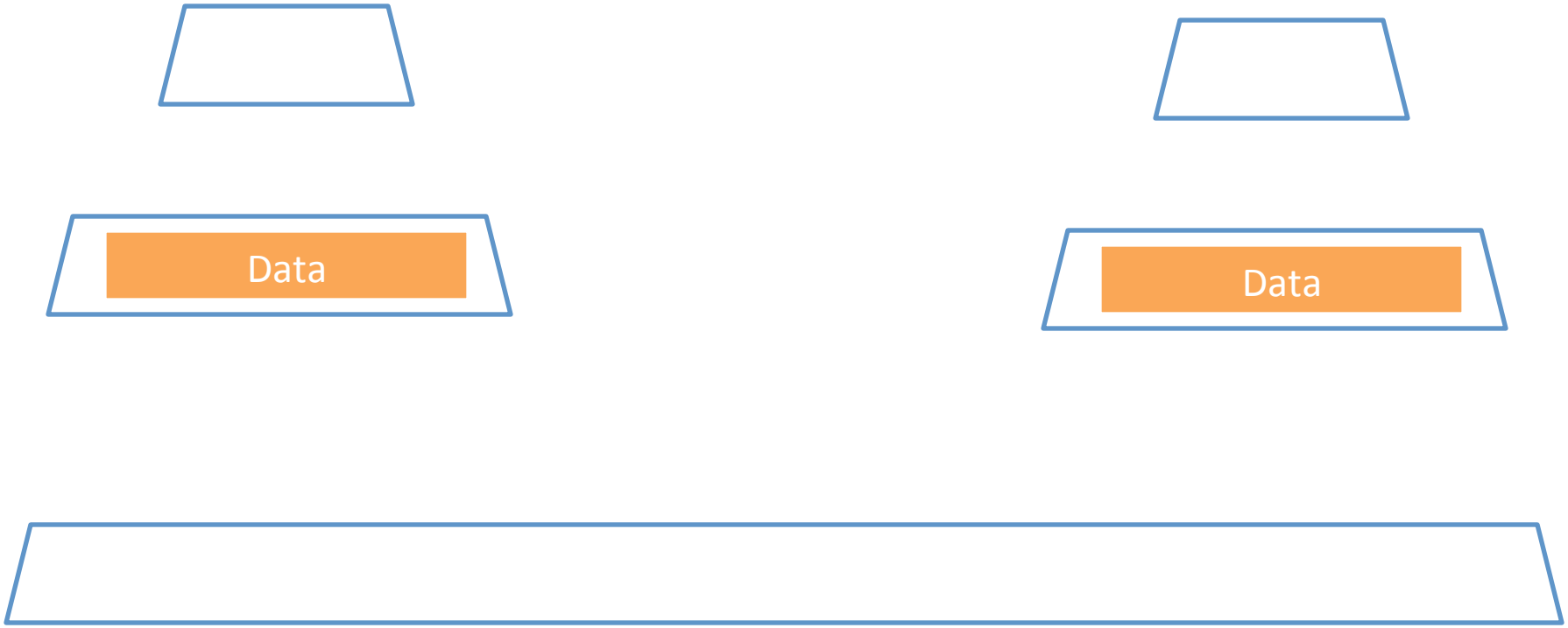


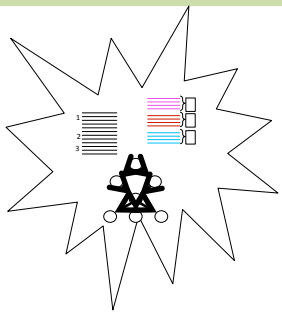




Sequoia

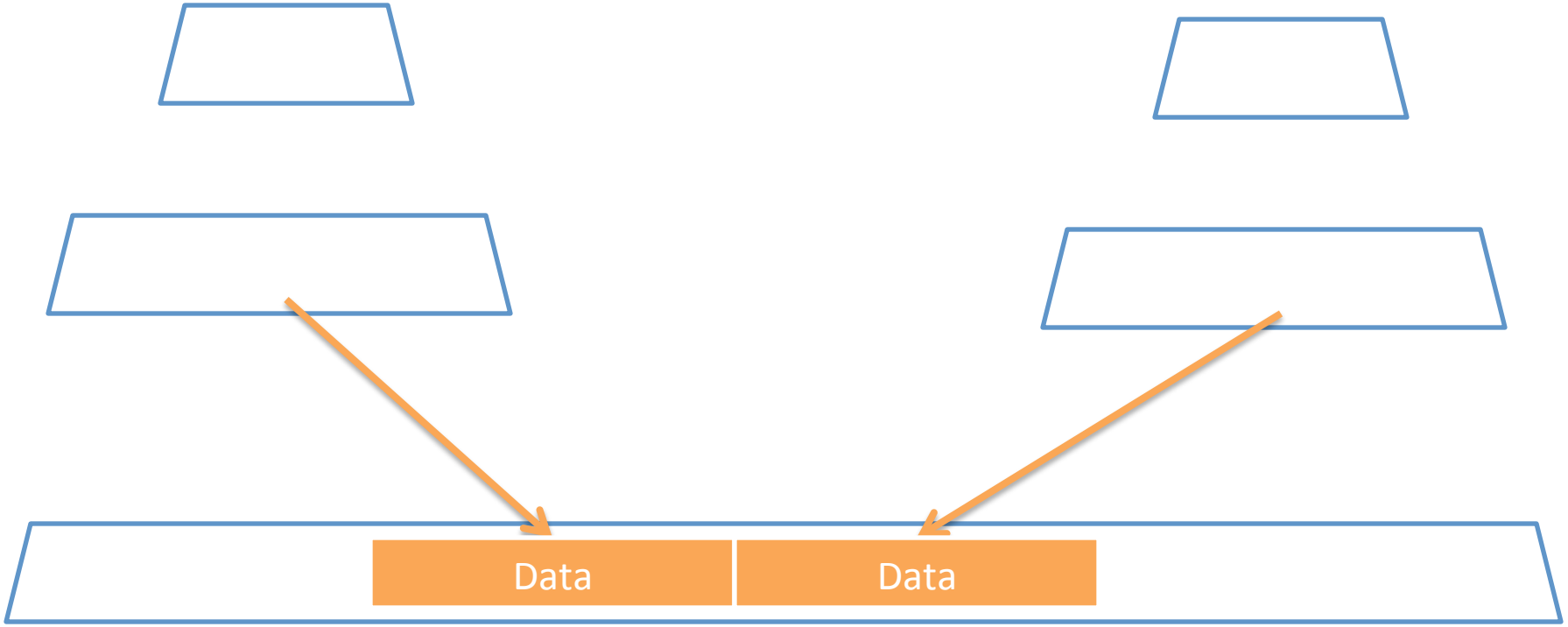
Realization of Distributed computation

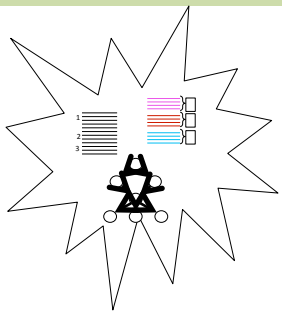




Sequoia

Realization of Distributed computation





Sequoia

Realization of Distributed computation