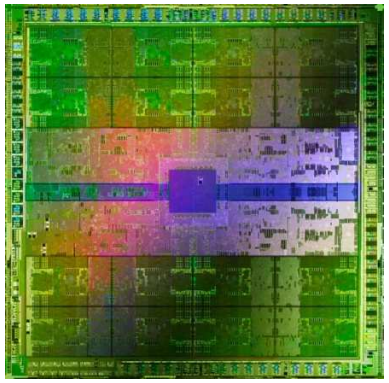
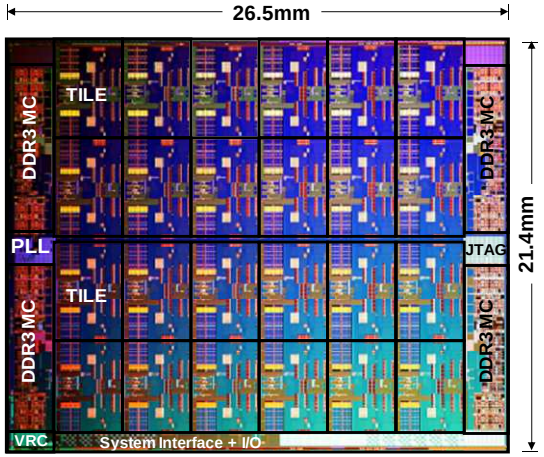
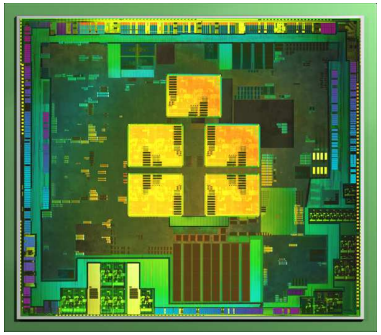


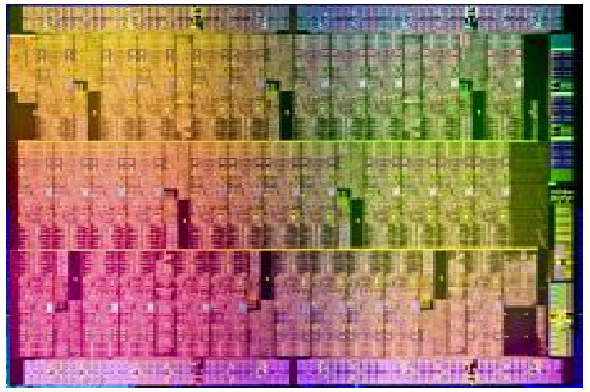
NVIDIA GTX 480 processor



Intel labs 48 core SCC processor



NVIDIA Tegra 3 (quad Arm Corex A9 cores + GPU)

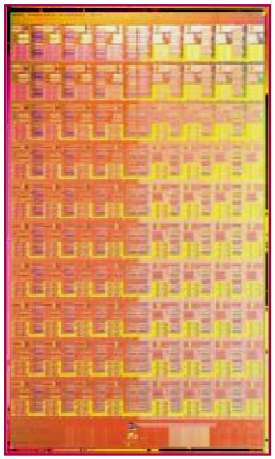


An Intel MIC processor

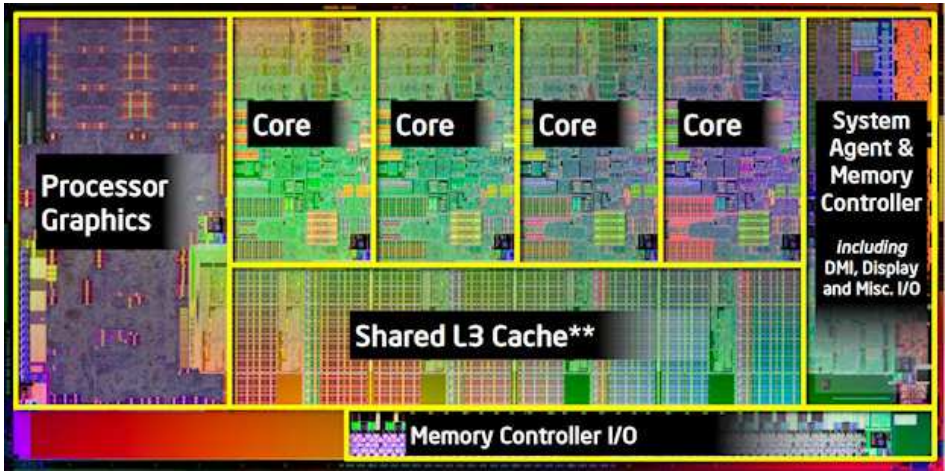
Parallel Computing: SW Architecture, design patterns and surviving the many core revolution

Part 1: introduction and background

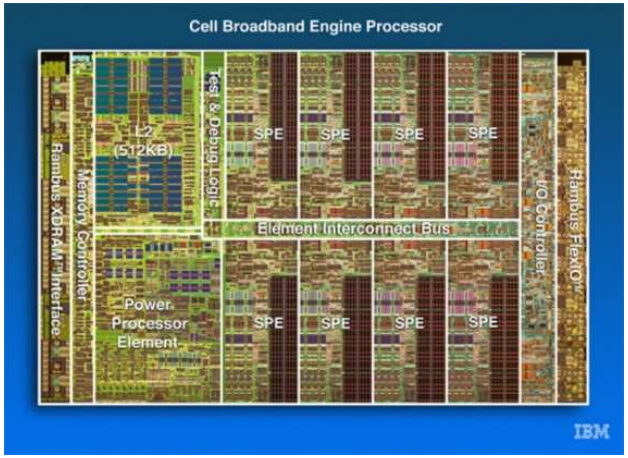
Tim Mattson (Intel Labs) and Michael Wrinn (Intel SSG)



Intel Labs 80 core Research processor



Intel "Sandybridge" processor



IBM Cell Broadband engine processor

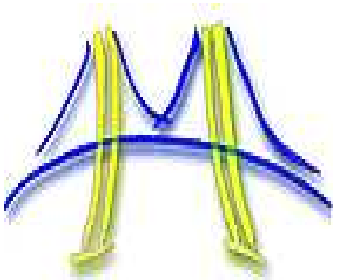
Other than the Intel lab's research processors. Die photos from UC Berkeley CS194 lecture notes

Third party names are the property of their owners

Disclaimer

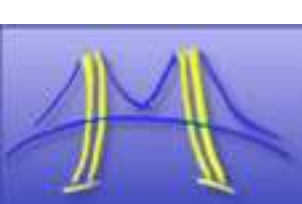
READ THIS … its very important

- The views expressed in this talk are those of the speakers and not their employer.
- This is an academic style talk and does not address details of any particular Intel product. You will learn nothing about Intel products from this presentation.
- This was a team effort, but if we say anything really stupid, it's our fault … don't blame our collaborators.



Slides marked with this symbol were produced-with Kurt Keutzer and his team for CS194 … A UC Berkeley course on Architecting parallel applications with Design Patterns.

Third party names are the property of their owners.



How NOT to write parallel programs

Legacy Code

Profiler

Performance profile

Not fast enough

Re-code with more threads

Deadlock !?!?

Re-code with fewer threads

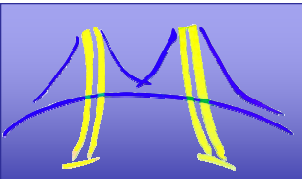
Non-Deterministic Concurrency Bugs

Re-code with 1 thread ?

Fast enough

Ship it

Who cares how fast it is if it doesn't work?

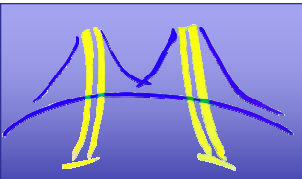


Why patterns? The premise:

PARLAB

- Difficulties in facing the challenge of developing parallel software are a symptom of underlying weakness in our abilities to:
  - Architect software
  - Re-use implementation approaches

- If the solution to the parallel programming doesn't have implications for all types of software design, then it's not a viable solution to parallel programming!

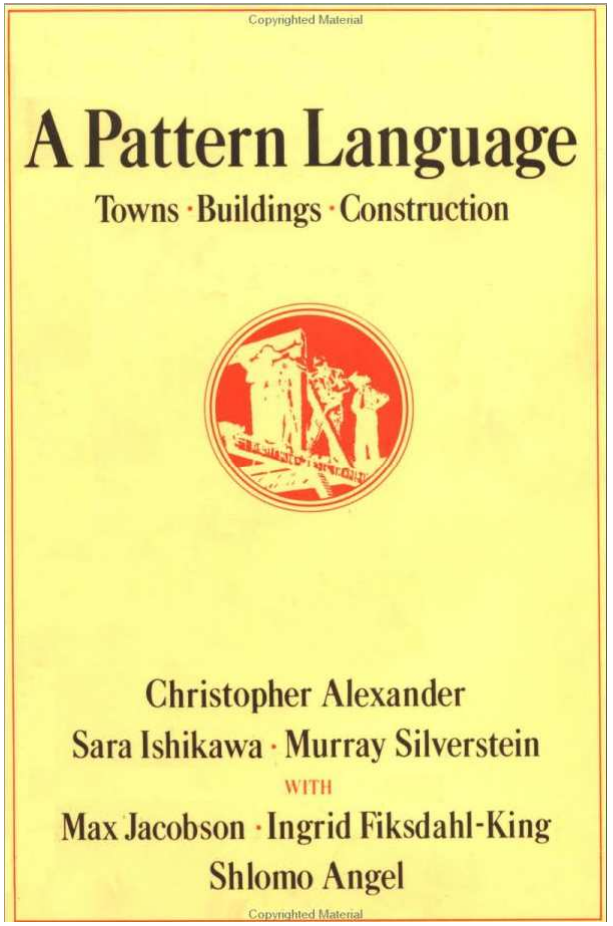


Outline

- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

Alexander's Pattern Language

- Christopher Alexander's approach to (civil) architecture:
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." *Page x, A Pattern Language,* Christopher Alexander
- Alexander's 253 (civil) architectural patterns range from the creation of cities (2. distribution of towns) to particular building problems (232. roof cap)
- A pattern language is an organized way of tackling an architectural problem using patterns
- Main limitation:
  - It's about civil not software architecture!!!



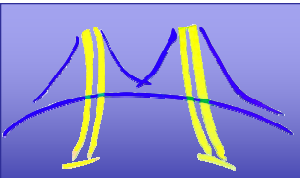
Family of Entrances (102)

- May be part of Circulation Realms (98).
- Conflict:
- When a person arrives in a complex of offices or services or workshops, or in a group of related houses, there is a good chance he will experience confusion unless the whole collection is laid out before him, so that he can see the entrance of the place where he is going.

Resolution:

Lay out the entrances to form a family. This means:

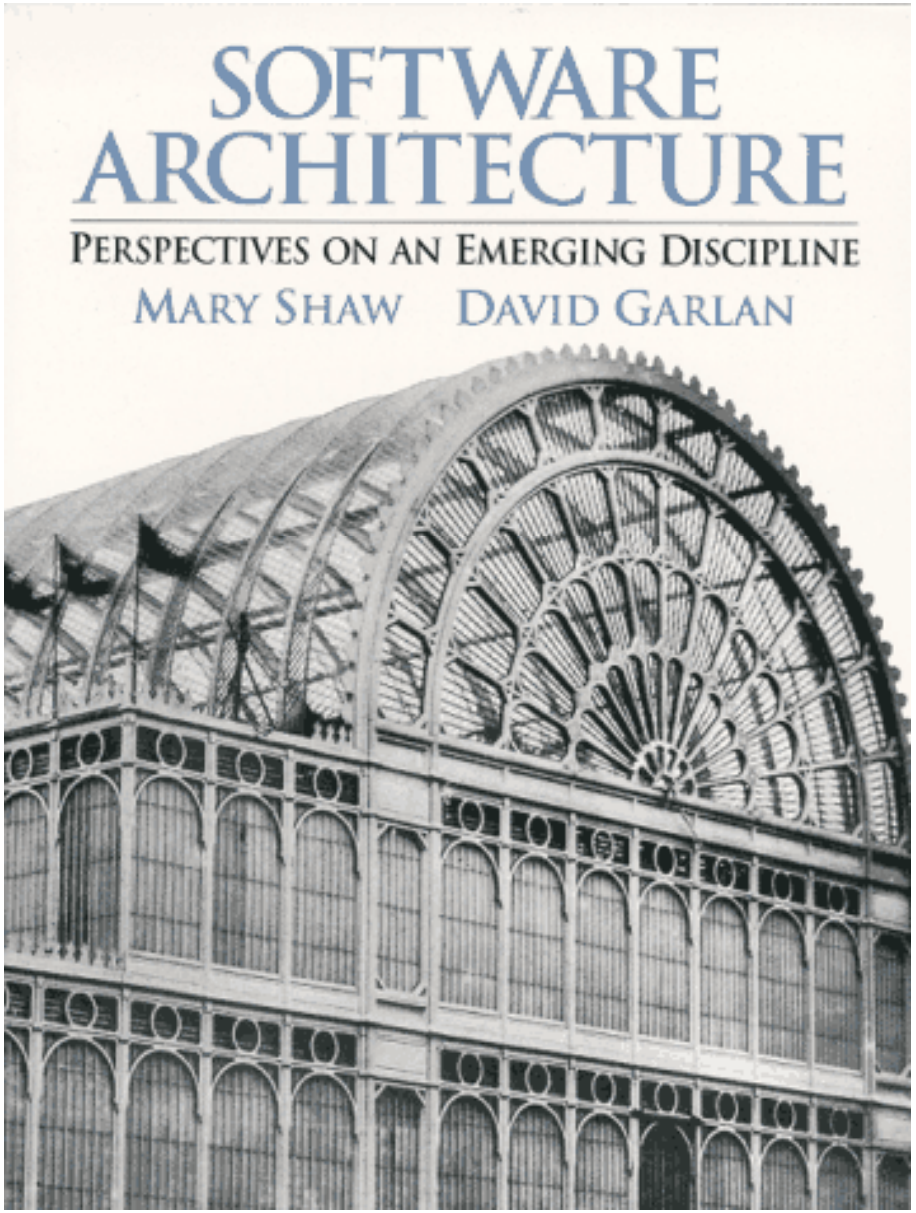
- 1) They form a group, are visible together, and each is visible from all the others.
- 2) They are all broadly similar, for instance all porches, or all gates in a wall, or all marked by a similar kind of doorway.
- May contain Main Entrance (110), Entrance Transition (112), Entrance Room (130), Reception Welcomes You (149).



Structural programming patterns

PA AS

- In order to create more complex software it is necessary to compose programming patterns
- For this purpose, it has been useful to induct a set of patterns known as “architectural styles”
- Examples:
  - pipe and filter
  - event based/event driven
  - layered
  - Agent and repository/blackboard
  - process control
  - Model-view-controller



Computational Patterns

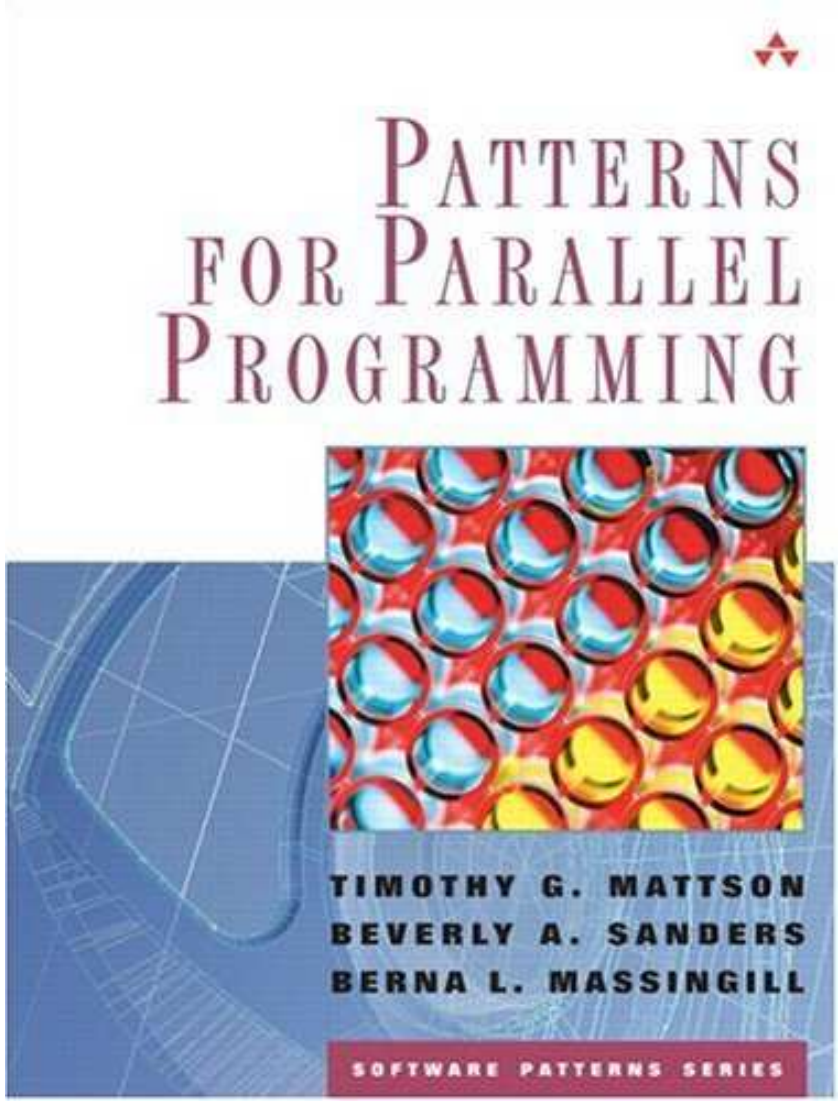
Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

Derived from the Dwarfs in "The Berkeley View" (Asanovic et al.)
Dwarfs form our key computational patterns

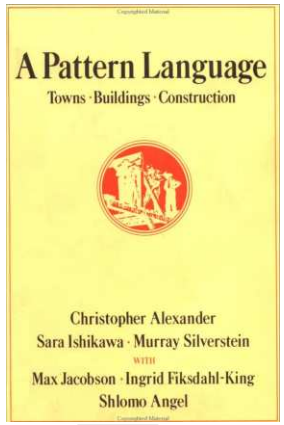
Patterns for Parallel Programming

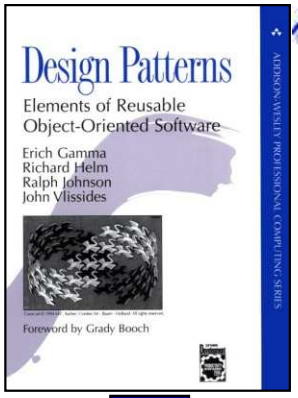
- PLPP is the first attempt to develop a complete *pattern language* for parallel software development.
- PLPP is a great model for a pattern language for parallel software
- PLPP mined scientific applications that utilize a monolithic application style
- PLPP doesn't help us much with horizontal composition

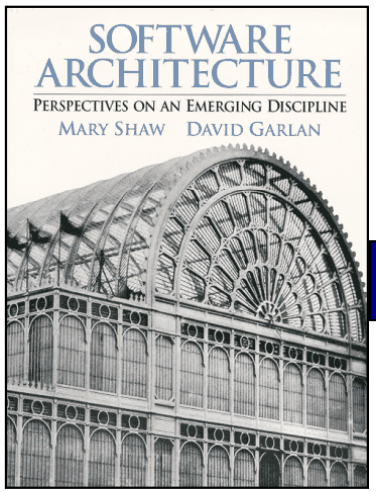
- Much more useful to us than: *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson & Vlissides, Addison-Wesley, 1995.

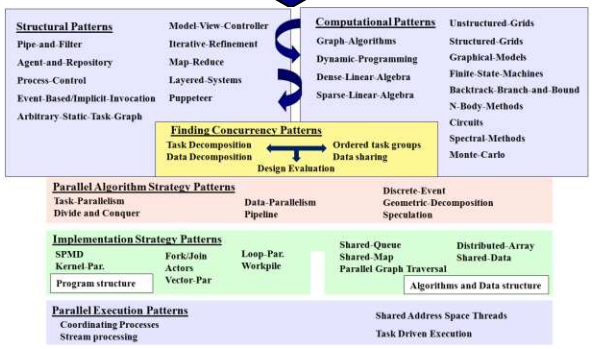


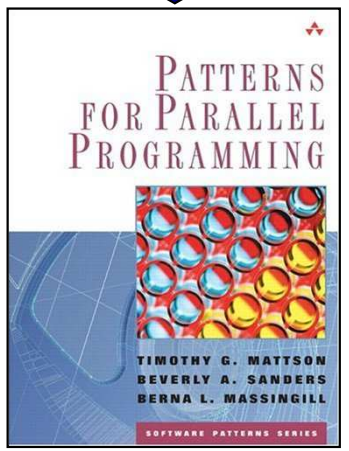
To get frameworks right … start with an understanding of software architecture



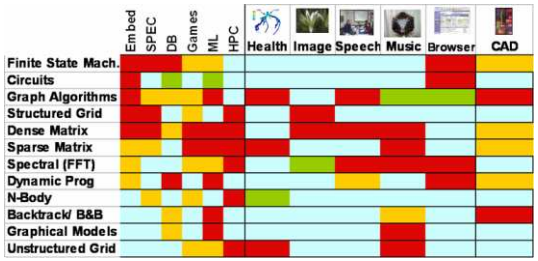






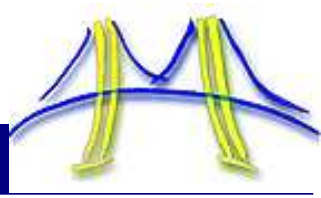


PLPP: Pattern language of Parallel Programming



13 dwarves

OPL Pattern Language (Keutzer & Mattson 2010)



Applications

Structural Patterns

- Pipe-and-Filter
- Agent-and-Repository
- Process-Control
- Event-Based/Implicit-Invocation
- Arbitrary-Static-Task-Graph
- Model-View-Controller
- Iterative-Refinement
- Map-Reduce
- Layered-Systems
- Puppeteer

Computational Patterns

- Graph-Algorithms
- Dynamic-Programming
- Dense-Linear-Algebra
- Sparse-Linear-Algebra
- Unstructured-Grids
- Structured-Grids
- Graphical-Models
- Finite-State-Machines
- Backtrack-Branch-and-Bound
- N-Body-Methods
- Circuits
- Spectral-Methods
- Monte-Carlo

Finding Concurrency Patterns

Task Decomposition ⟷ Ordered task groups
Data Decomposition ⟷ Data sharing
Design Evaluation

Parallel Algorithm Strategy Patterns

- Task-Parallelism
- Divide and Conquer
- Data-Parallelism
- Pipeline
- Discrete-Event
- Geometric-Decomposition
- Speculation

Implementation Strategy Patterns

Program structure:
- SPMD
- Kernel-Par.
- Fork/Join
- Actors
- Vector-Par
- Loop-Par.
- Workpile

Algorithms and Data structure:
- Shared-Queue
- Shared-Map
- Parallel Graph Traversal
- Distributed-Array
- Shared-Data

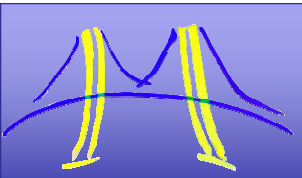
Parallel Execution Patterns

- Coordinating Processes
- Stream processing
- Shared Address Space Threads
- Task Driven Execution

Concurrency Foundation constructs (not expressed as patterns)

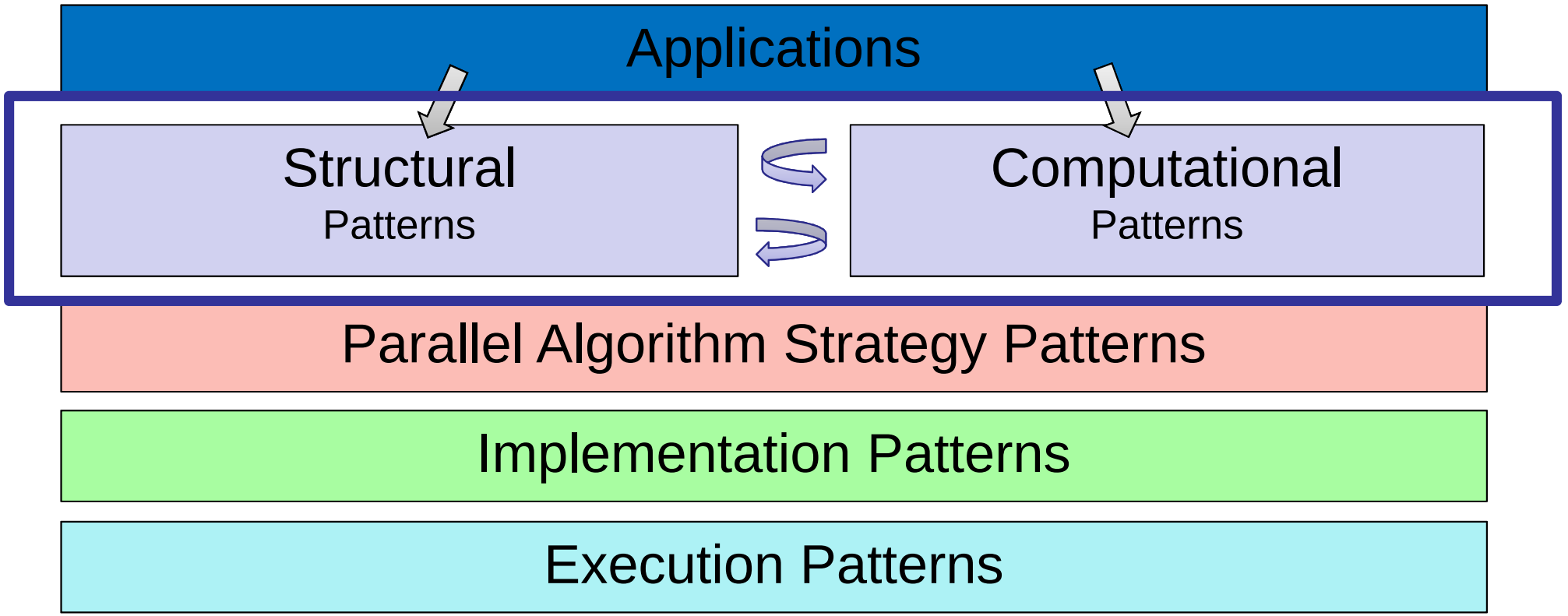
Thread/proc management | Communication | Synchronization

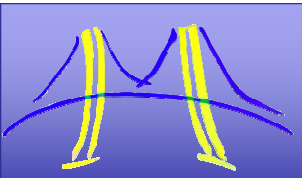
Source: Keutzer and Mattson Intel Technology Journal, 2010



Our Pattern Language

PAMAS





Outline

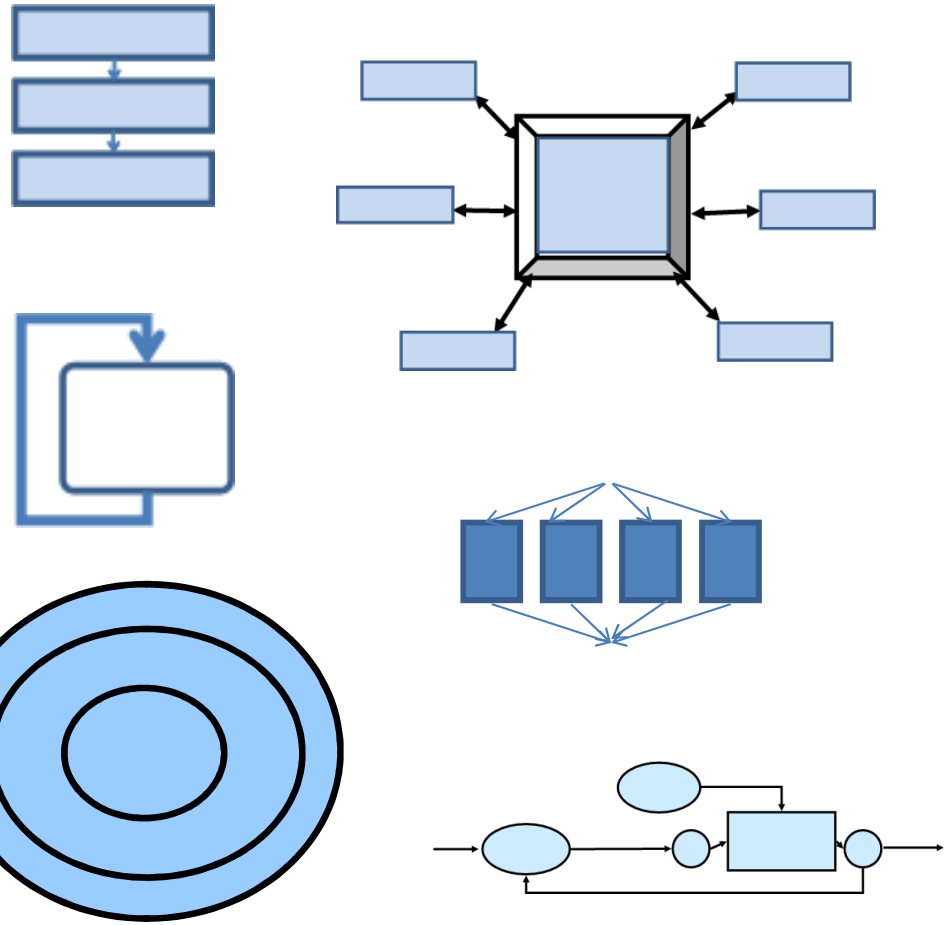
PARLAB

- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software: structure & computation
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

Identify the SW Structure

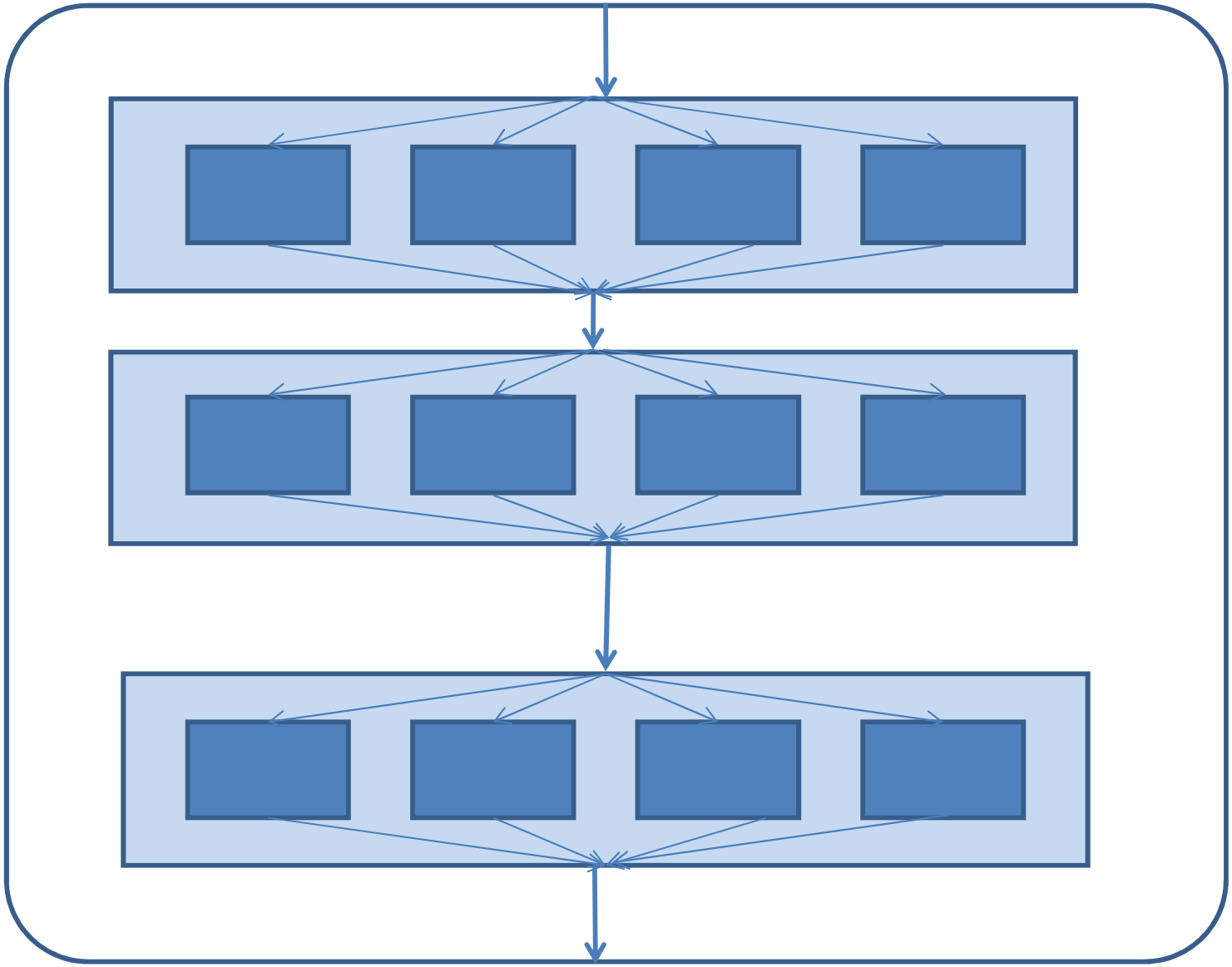
Structural Patterns

- Pipe-and-Filter
- Agent-and-Repository
- Process-Control
- Event-Based/Implicit-Invocation
- Puppeteer
- Model-View-Controller
- Iterative-Refinement
- Map-Reduce
- Layered-Systems
- Arbitrary-Static-Task-Graph



These define the software structure but *do not describe* what is computed

Analogy: Layout of Factory Plant

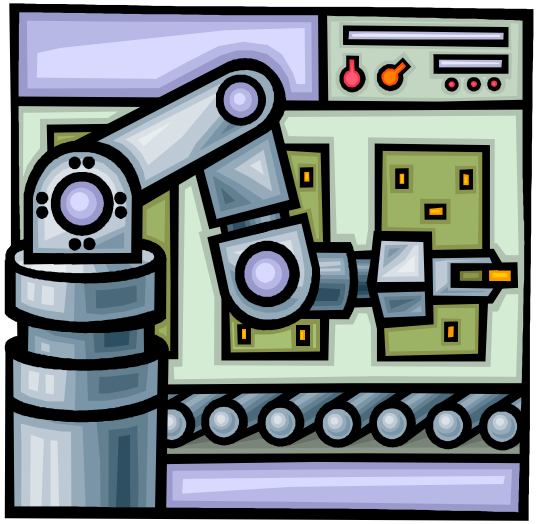


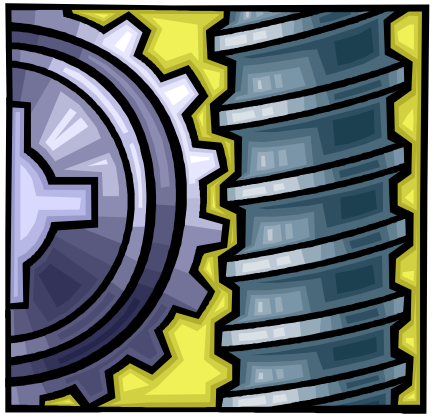
Identify Key Computations

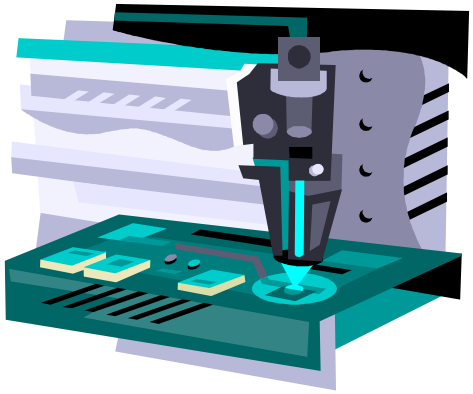
Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

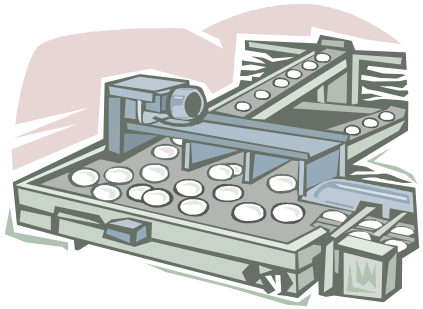
These define the key computations, but *do not describe* how they are implemented

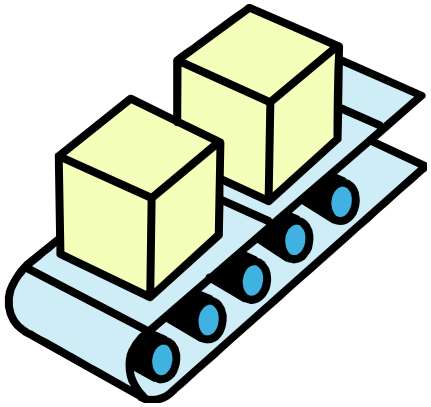
Analogy: Machinery of the Factory

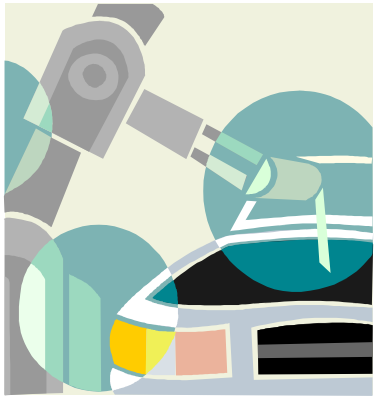




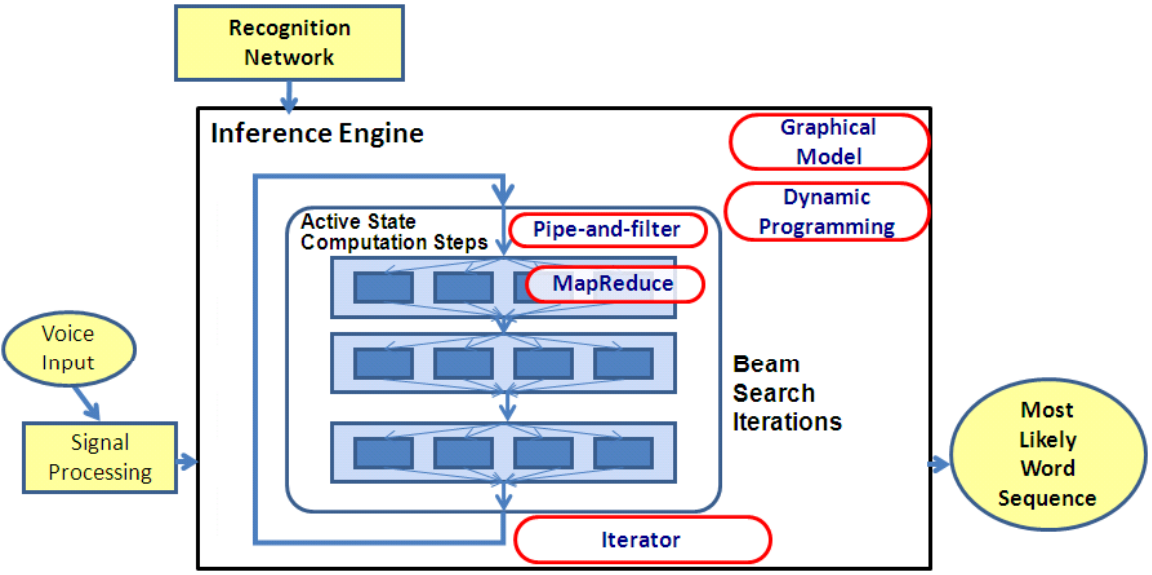


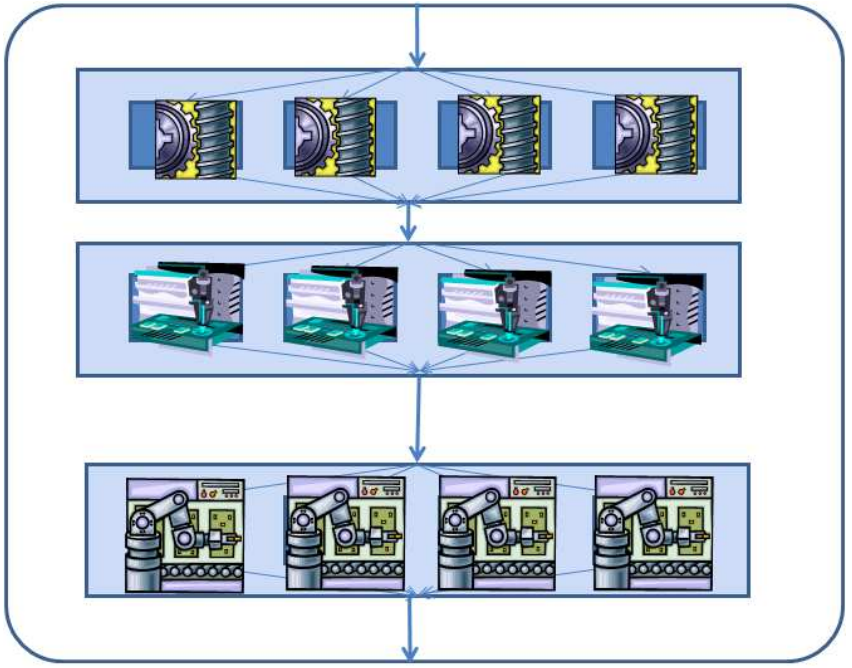






Architecting the Whole Application

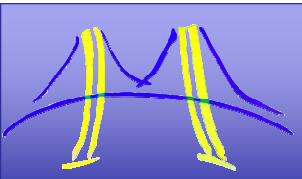




- SW Architecture of Large-Vocabulary Continuous Speech Recognition

Analogous to the design of an entire manufacturing plant

- Raises appropriate issues like scheduling, latency, throughput, workflow, resource management, capacity etc.



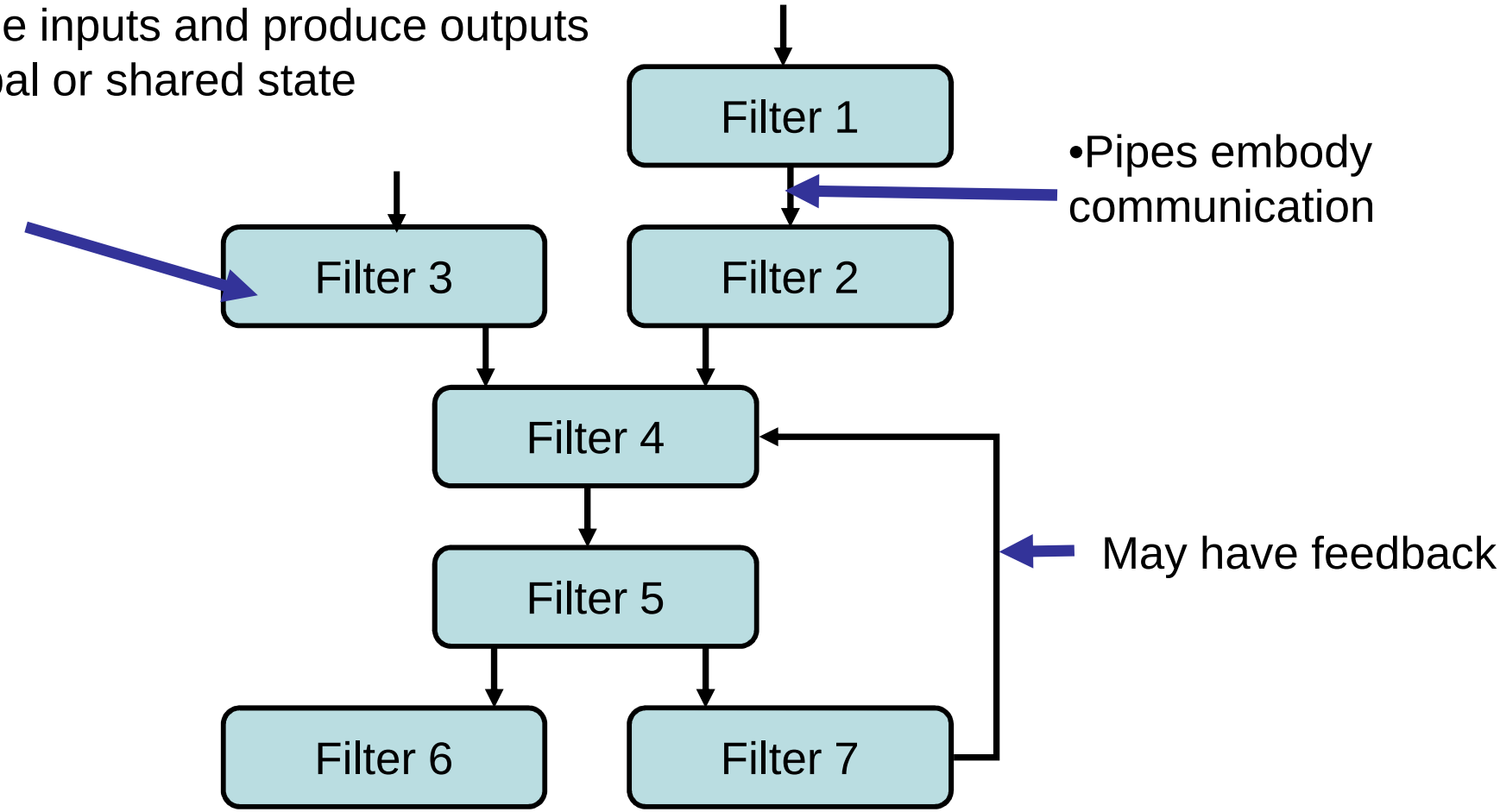
Outline

PARLAB

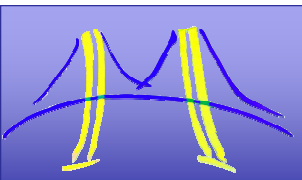
- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
- Structural Patterns – 2 examples (of 10)
  - Pipe and filter
  - Map Reduce
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

Structural Pattern: Pipe and Filter

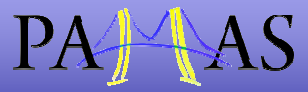
- Filters embody computation
- Only see inputs and produce outputs
- No global or shared state



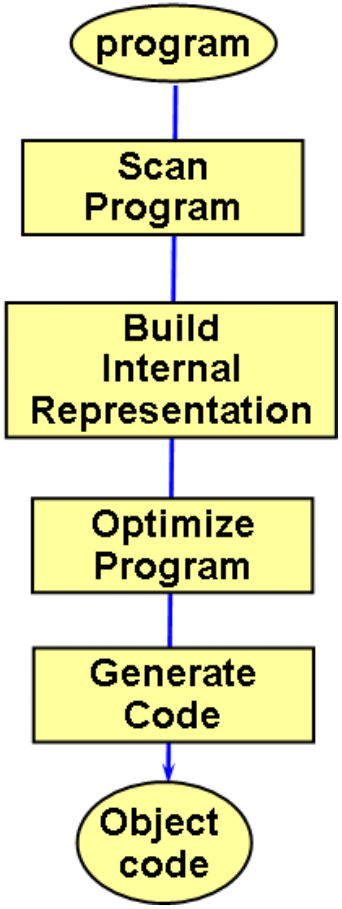
Examples?



Examples of pipe and filter



- Almost every large software program has a pipe and filter structure at the highest level



Compiler

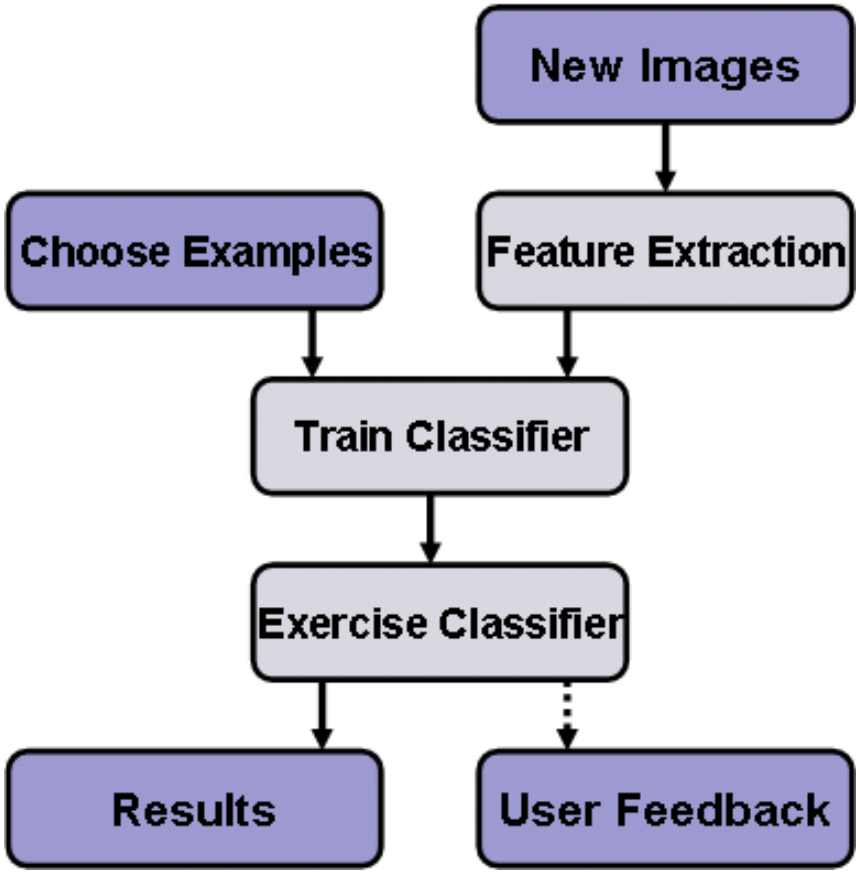
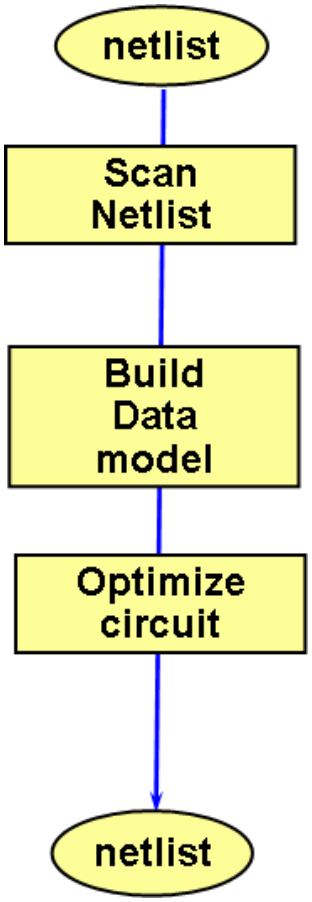
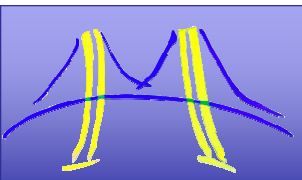


Image Retrieval System



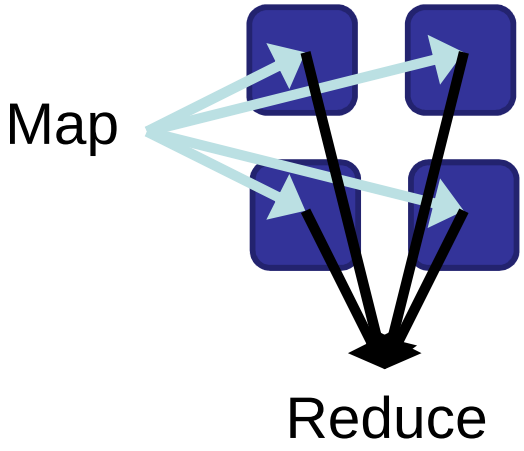
Logic optimizer

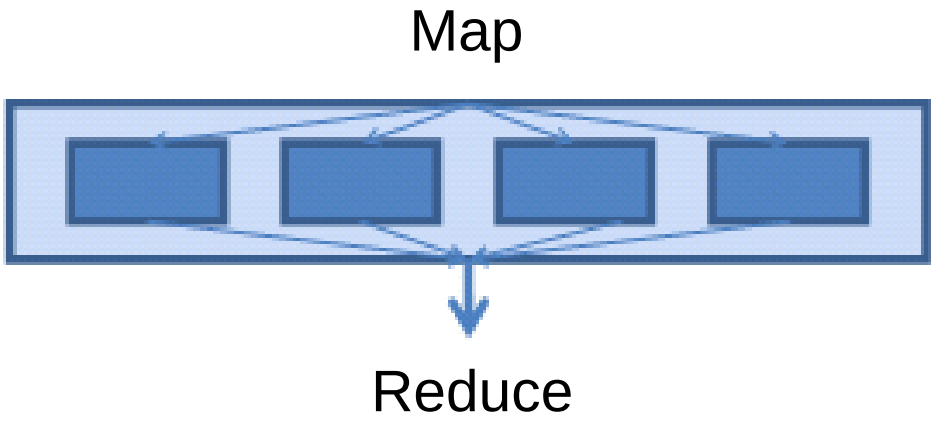


Structural Pattern: MapReduce

PAMAS

- To us, it means
  - A map stage, where data is mapped onto independent computations
  - A reduce stage, where the results of the map stage are summarized (i.e. reduced)

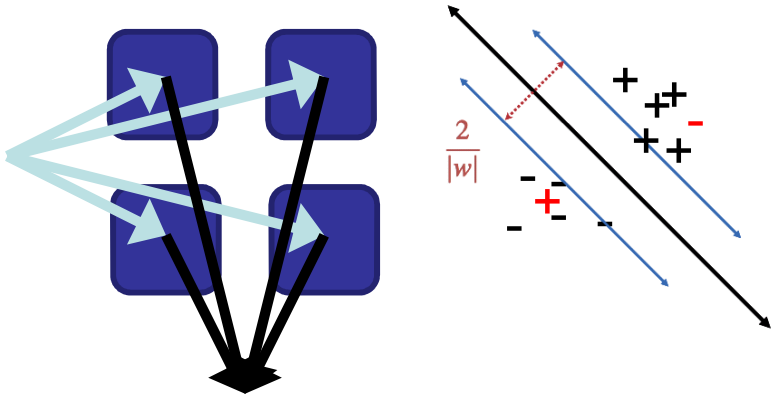




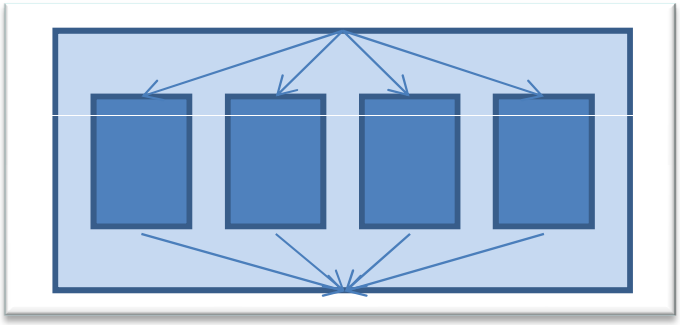
Examples?

Examples of Map Reduce

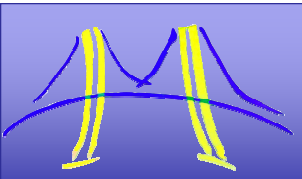
- General structure:
- Map a computation across distributed data sets
- Reduce the results to find the best/(worst), maxima/(minima)



Support-vector machines (ML)
• Map to evaluate distance from the frontier
• Reduce to find the greatest outlier from the frontier



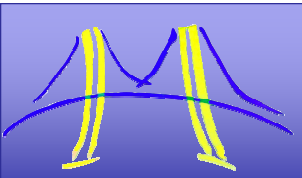
Speech recognition
• Map HMM computation to evaluate word match
• Reduce to find the most-likely word sequences



Outline

PATTERNS

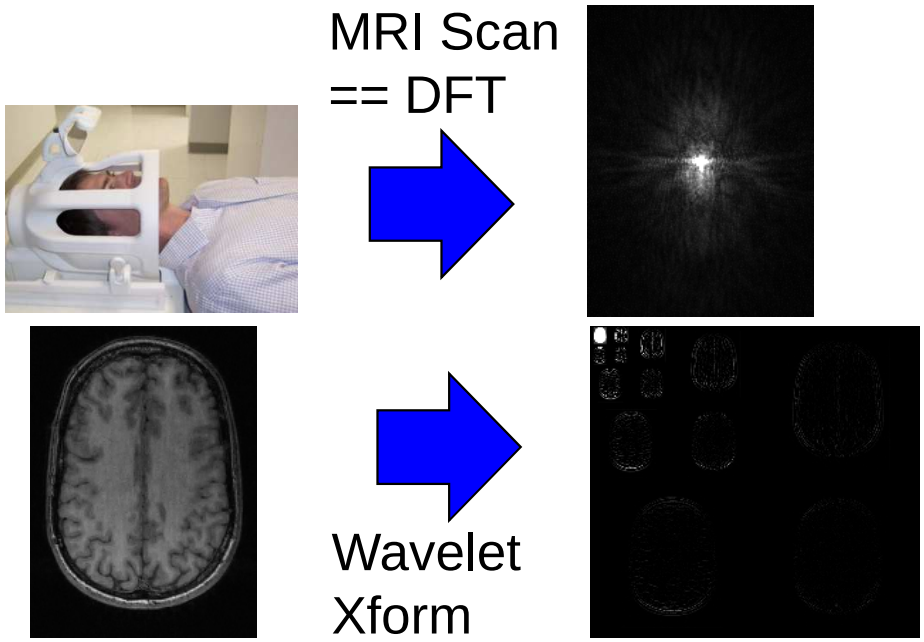
- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns – 1 example (of 13)
  - Spectral Methods
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

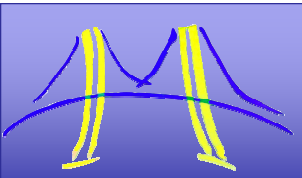


Computational Pattern: Spectral Methods

PARLAS

- "Spectral Methods" are a broad class of numerical algorithms for solving PDEs, but notions of Spectral Analysis (i.e. convenient changes of basis) are important in every application area
- In Magnetic Resonance Imaging (MRI), images are collected in "k-space" -- i.e. an MRI scan produces a Fourier Domain image
- Fourier and Wavelet representations are different Spectral analyses that expose different properties of images convenient for solving our problems





Spectral Methods Pattern: Fast Transforms

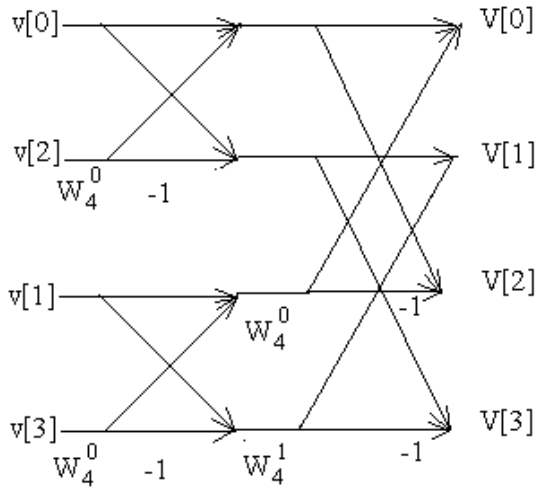
PARLAB

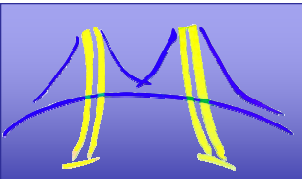
- Spectral Methods rely on represention of data in "convenient" bases that produce working, computationally *feasible* algorithms
- Changing a basis is, in general, an $O(N^2)$ matrix-vector multiplication. The matrices representing "convenient" bases factor into O(N log N) fast transforms!

$$y_k = \sum_{j=0}^{N-1} x_j \ \omega^{jk}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$$F_n \ x = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} \ x_{even} \\ F_{n/2} \ x_{odd} \end{bmatrix}$$

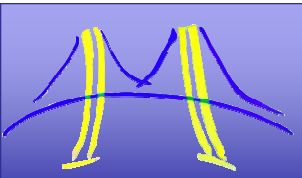




Outline

PA AS

- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

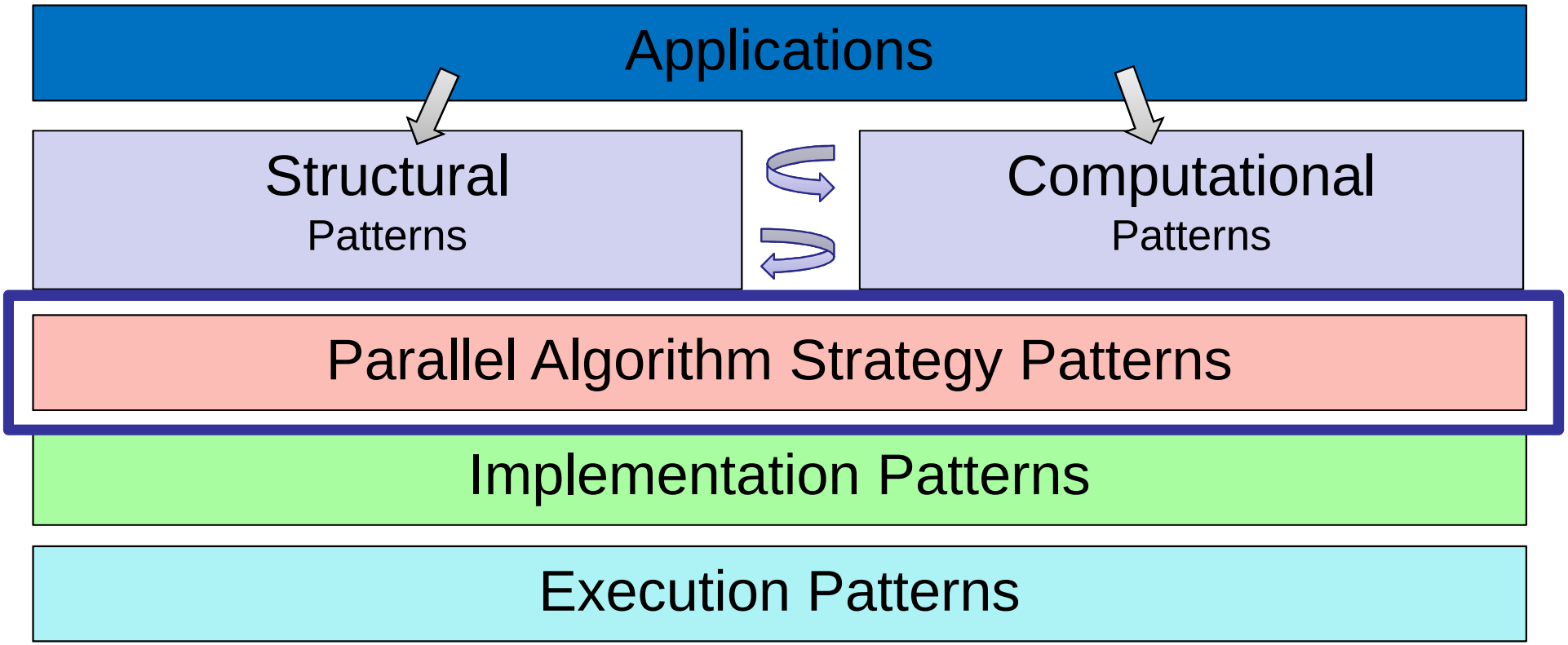


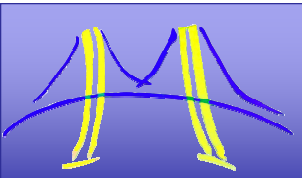
Structural Patterns and Parallelism

PA AS

- Structural patterns are “above the concerns of parallelism” – they may be used for either serial or parallel implementations
- When we attempt to parallelize a software architecture, the description in structural patterns gives us only a modest amount of parallelism
  - Map reduce may be the only exception among the structural patterns
- Therefore we need a new perspective on parallelizing the architecture – i.e. new patterns

OPL: Parallel Algorithm Strategy





Parallel Algorithm Strategy Patterns

PAMAS

These patterns define high-level strategies to exploit concurrency within a computation for execution on a parallel computer.

They address the different ways concurrency is naturally expressed within a problem/application.

How does the software architecture map onto parallel algorithms?

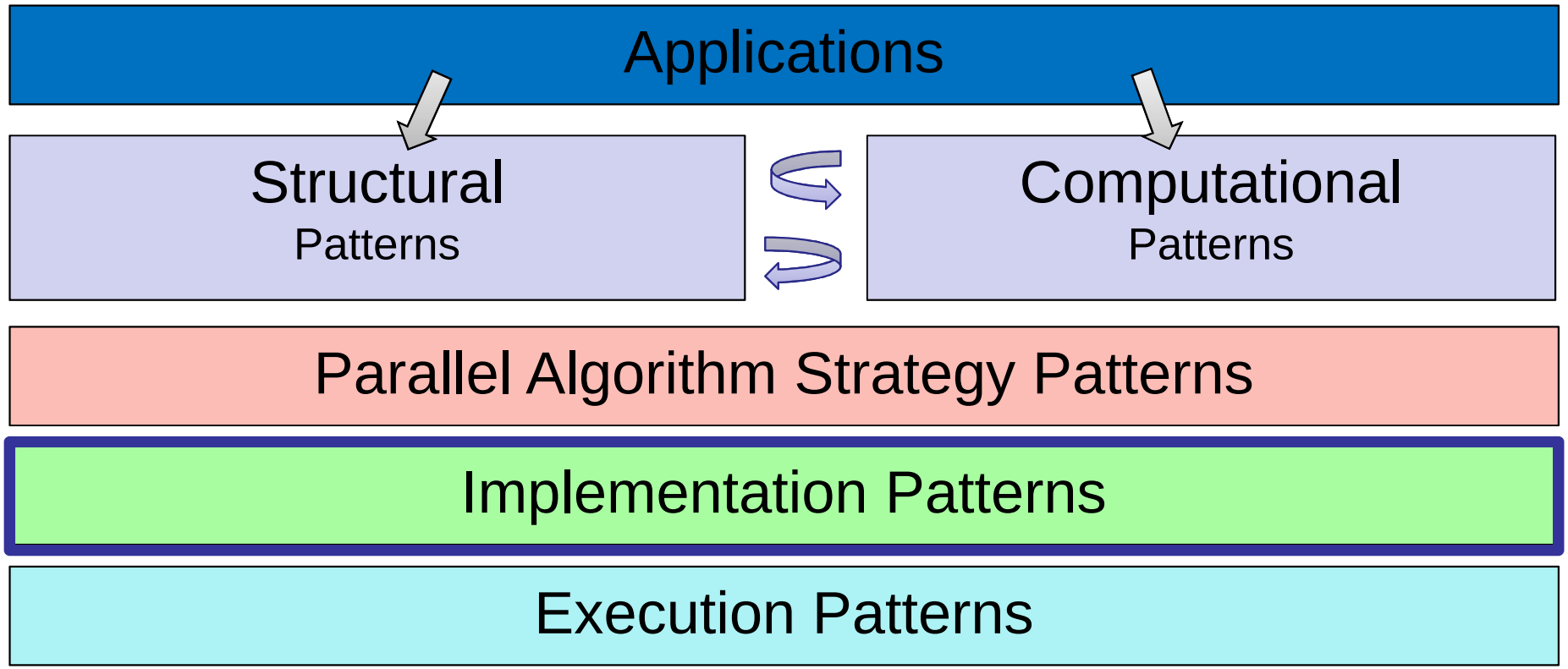
Algorithm Strategy Patterns

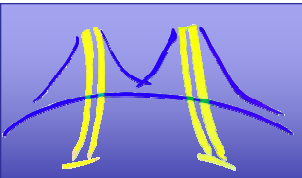
Task Parallelism
Recursive splitting

Data Parallelism
Pipeline

Discrete Event
Geometric Decomposition
Speculation

OPL: Parallel Algorithm Strategy





Implementation Strategy Patterns

PAMAS

These are the structures that are realized in source code to support (a) how the program itself is organized and (b) common data structures specific to parallel programming.

How do parallel algorithms map onto source code in a parallel programming language?

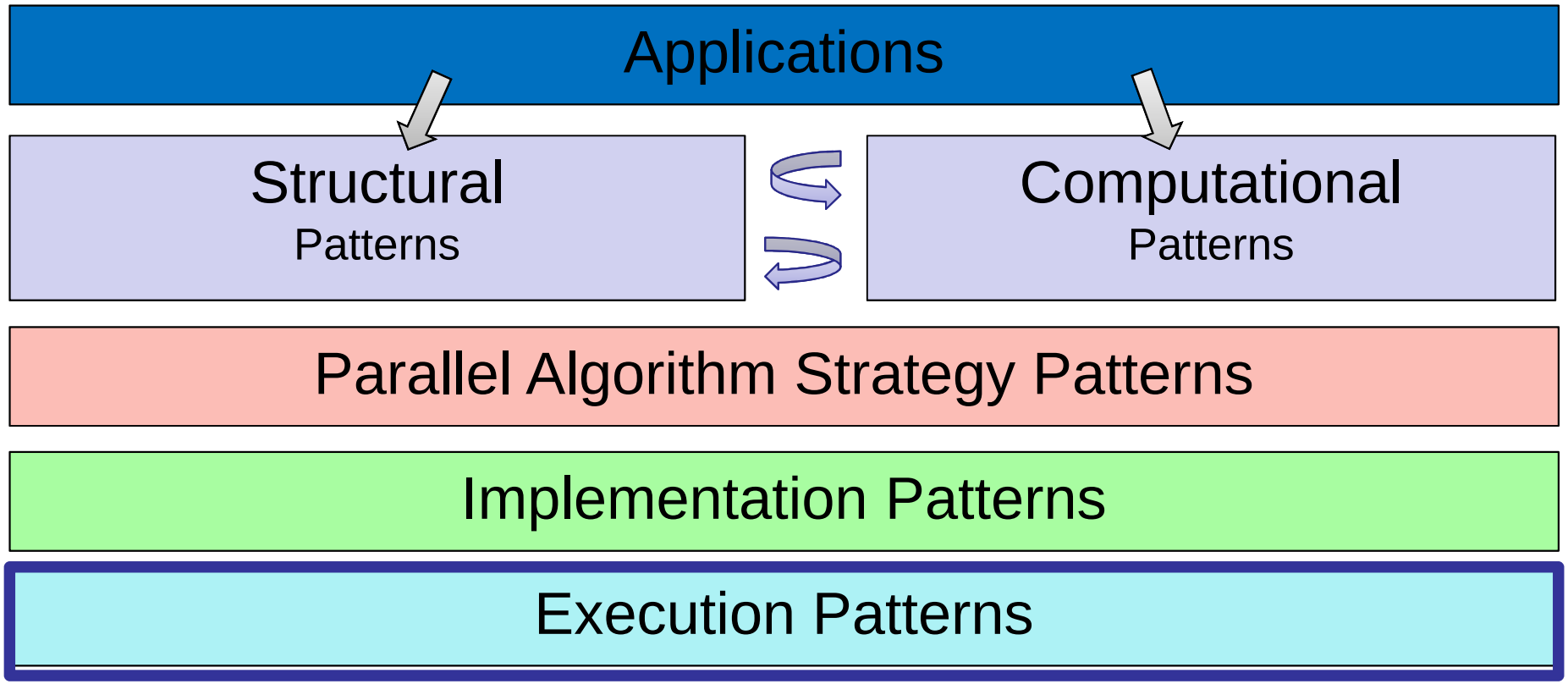
Implementation Strategy Patterns

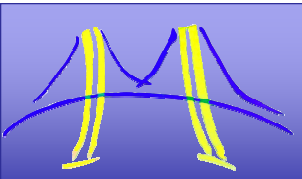
SPMD	Master/worker	Actors	Shared-Queue	Distributed-Array
Strict data par	Task-Queue	BSP	Shared Hash Table	Shared-Data
Fork/Join	Graph partitioning			
Loop-Par.				

Program structure

Data structure

OPL: Parallel Algorithm Strategy





Parallel Execution Patterns

PAMAS

These are the approaches often embodied in a runtime system that supports the execution of a parallel program.

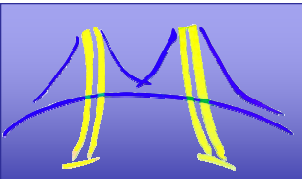
How is the source code realized as an executing program running on the target parallel processor?

Parallel Execution Patterns

MIMD SIMD	Thread Pool Task Graph	Task-Graph Data Flow Digital Circuits	Msg Pass Collective Comm Mutual exclusion	Pt-2-pt sync Collective sync Trans Mem

Advancing “program counters”

Coordination

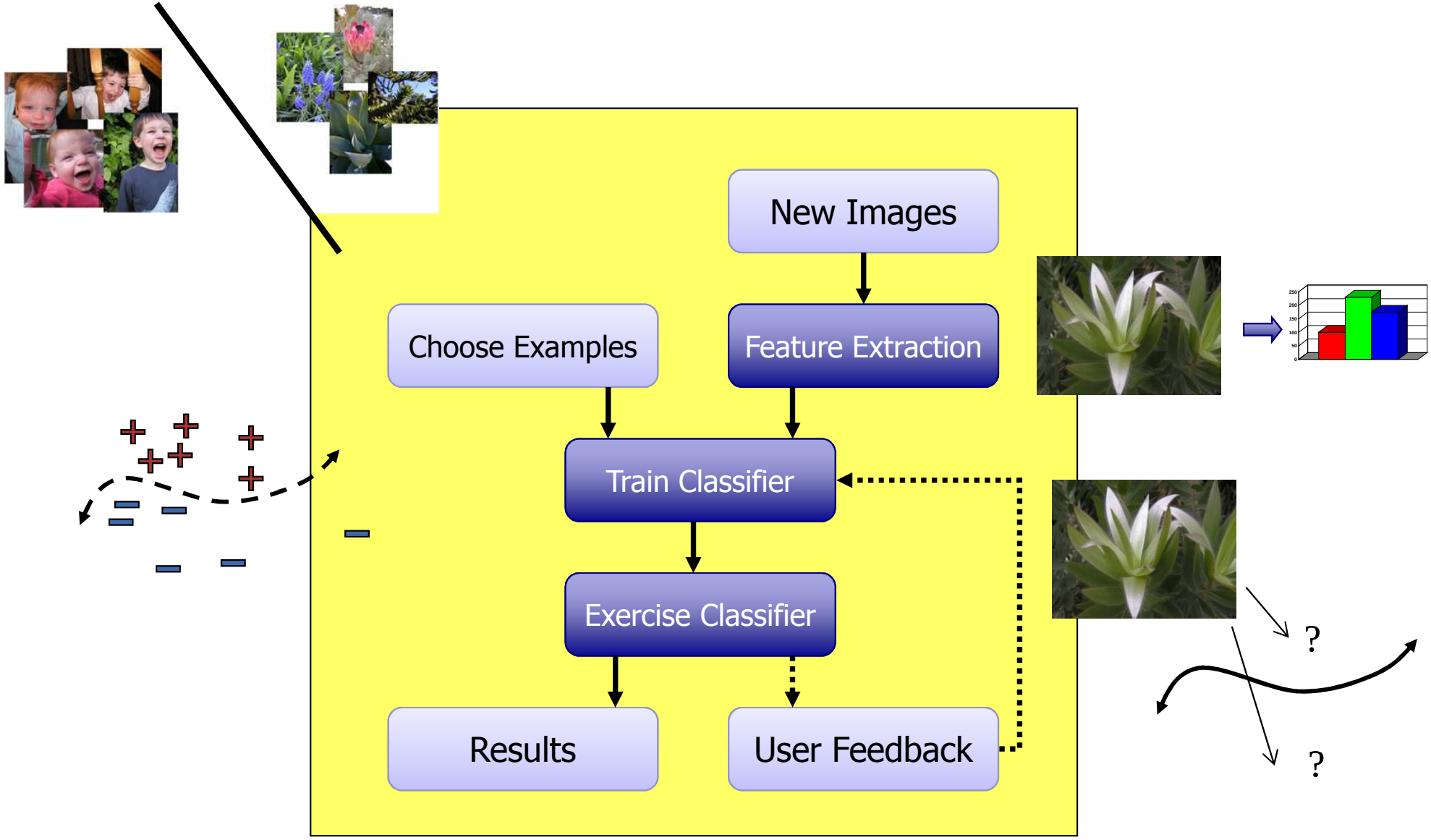


Outline

PAMAS

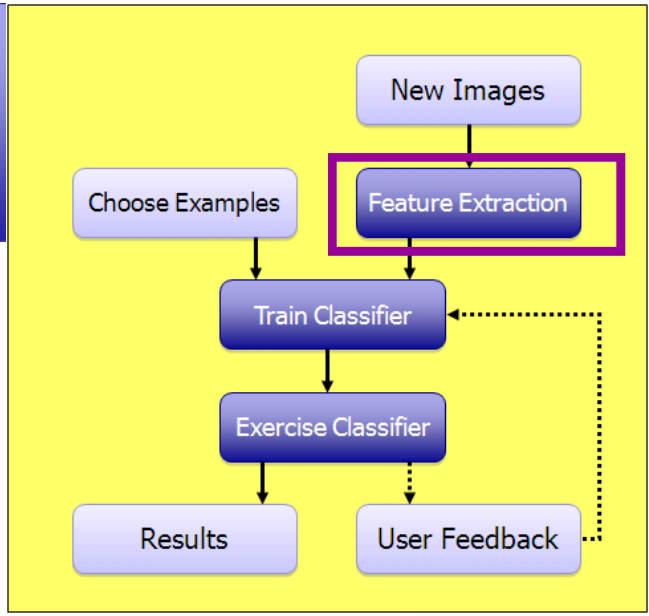
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - SVM classifier
- Summary

Support Vector Machine Classifier

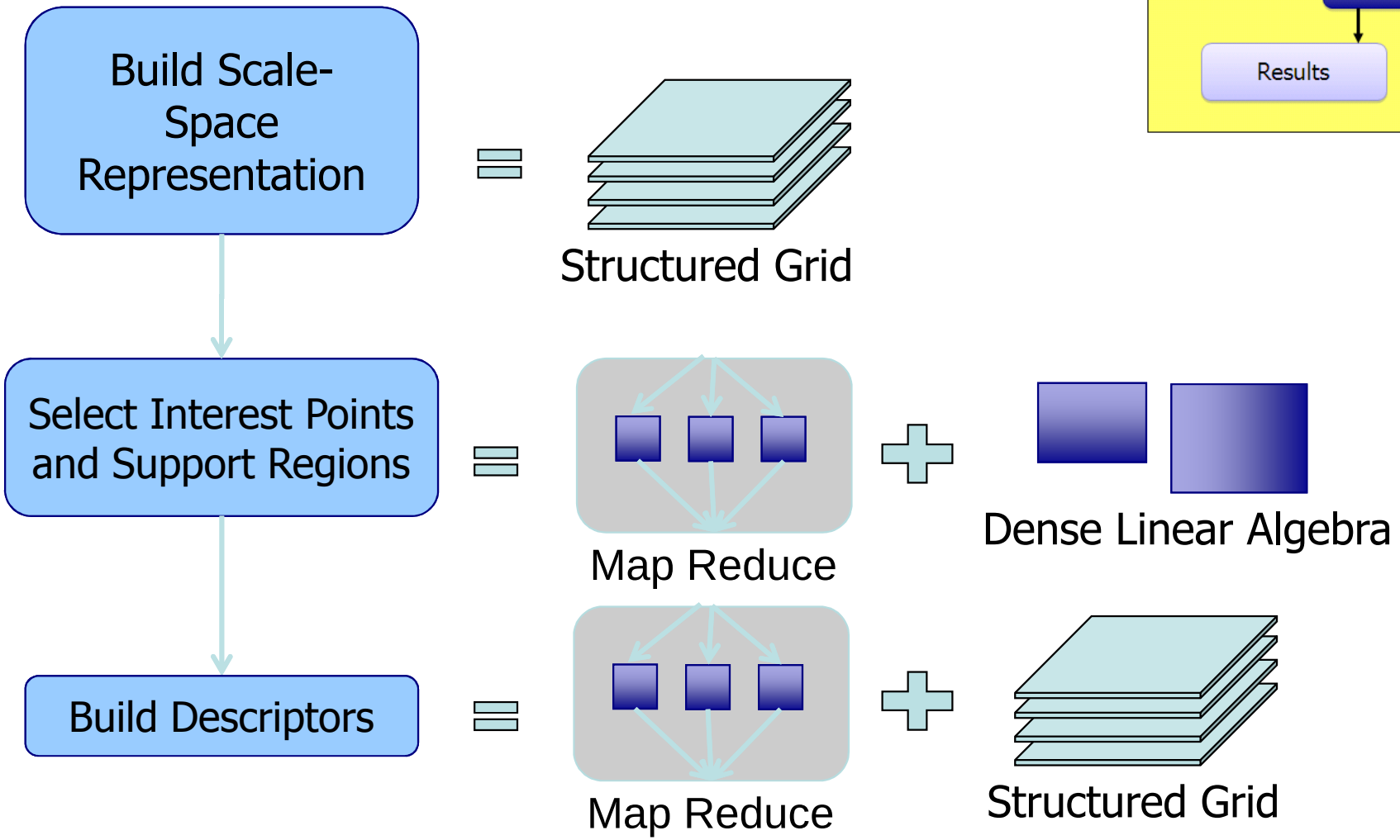


Catanzaro, Sundaram, Keutzer, “Fast SVM Training and Classification on Graphics Processors”, ICML 2008

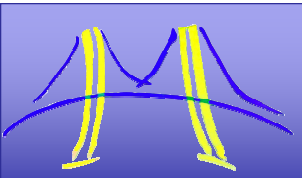
Feature Extraction



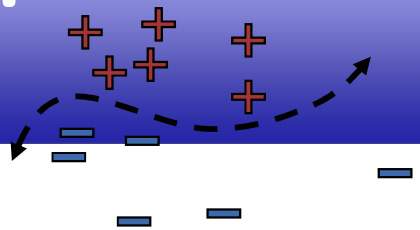
- Image is reduced to a set of low-dimensional feature vectors

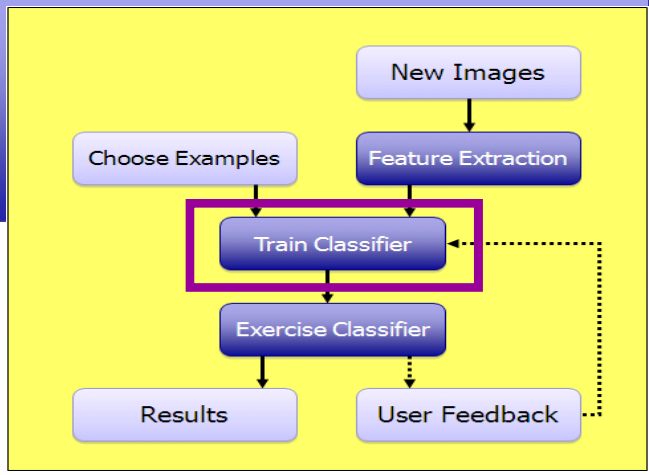


"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09



Train Classifier: SVM Training



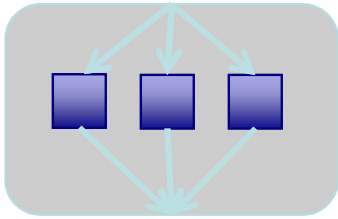


Train Classifier

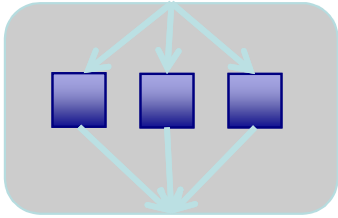
Update Optimality Conditions

Select Working Set, Solve QP

iterate



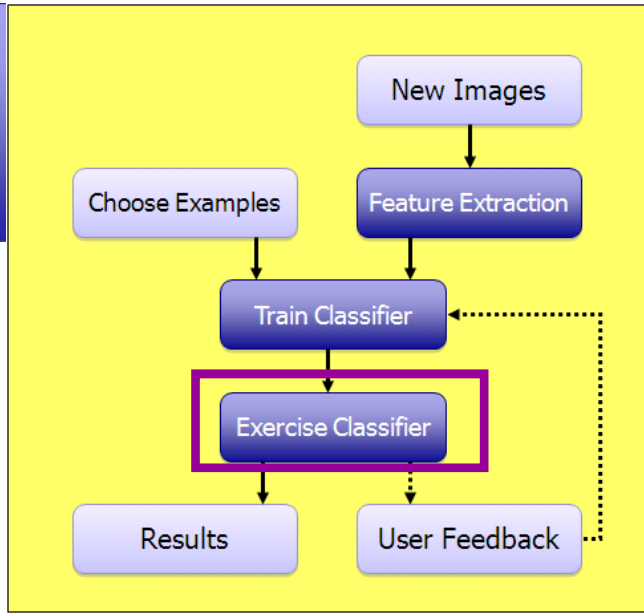
MapReduce



MapReduce

Iterator Pattern

Exercise Classifier : SVM Classification



Test Data

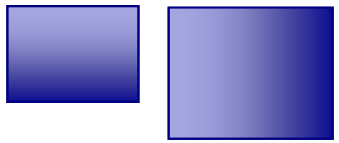
SV

Compute dot products

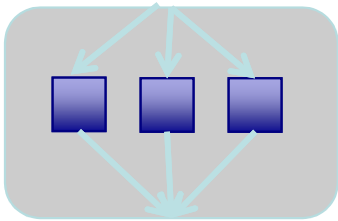
Compute Kernel values, sum & scale

Output

Exercise Classifier

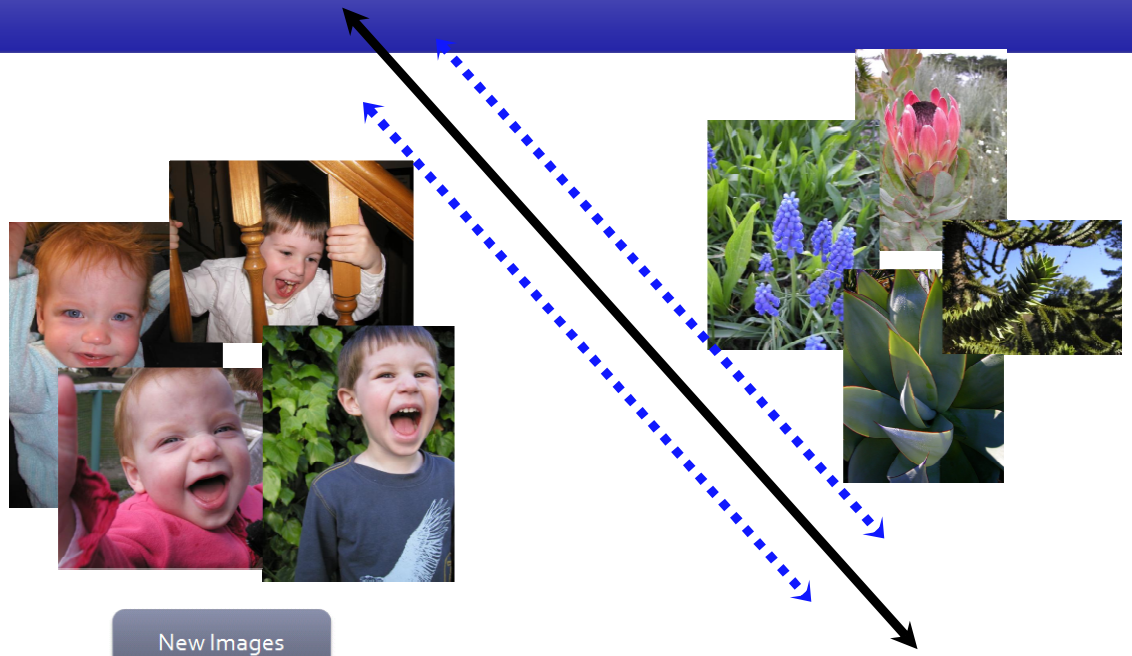


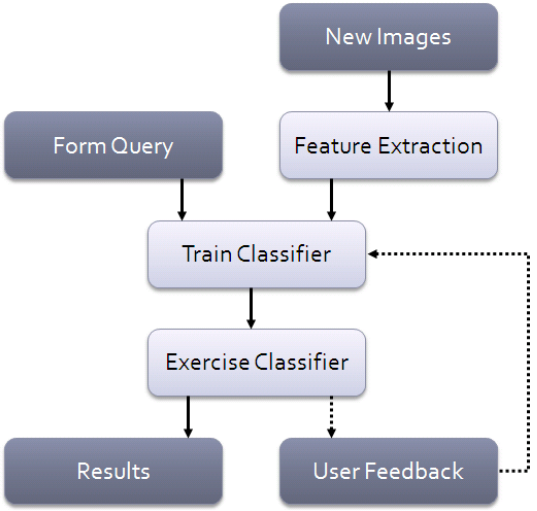
Dense Linear Algebra



MapReduce

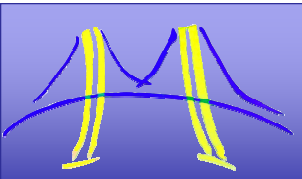
Support-Vector Machine Mini-Framework





- Support-Vector Machine Framework used to achieve:
  - 9-35x speedup for training
  - 81-138x for classification
- 1100 downloads since release

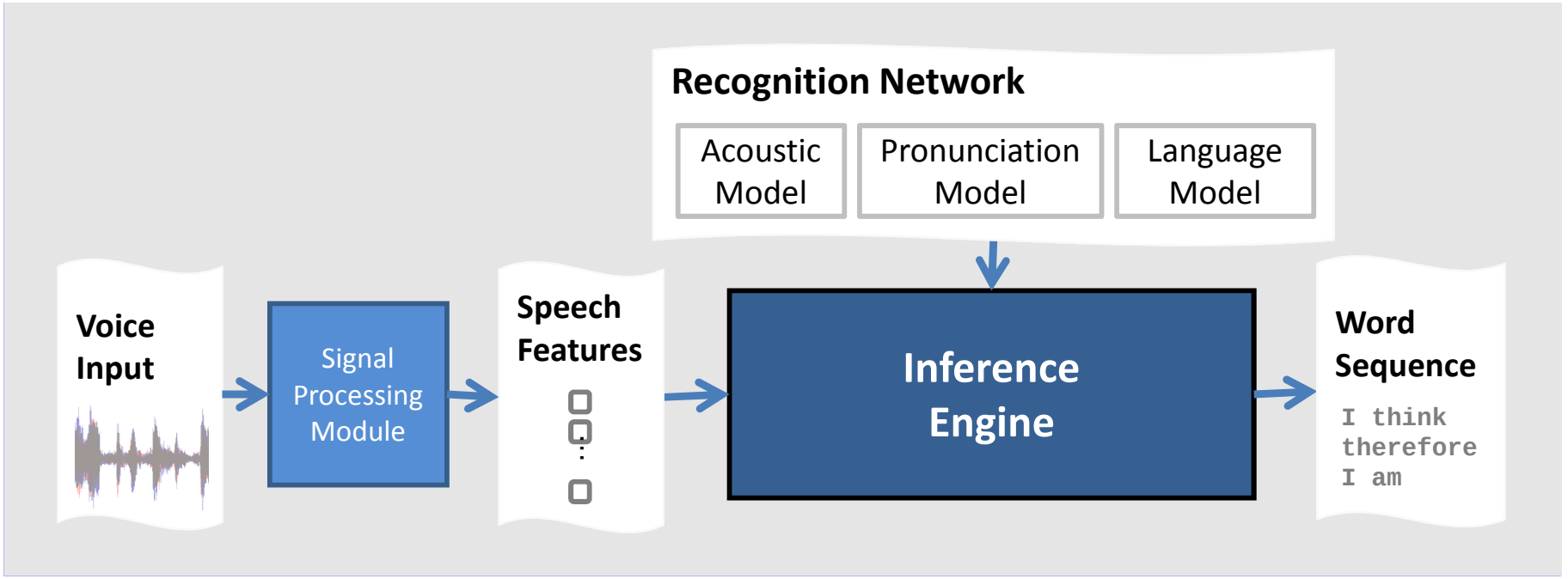
Fast support vector machine training and classification , Catanzaro, Sundaram, Keutzer, International Conference on Machine Learning 2008



Outline

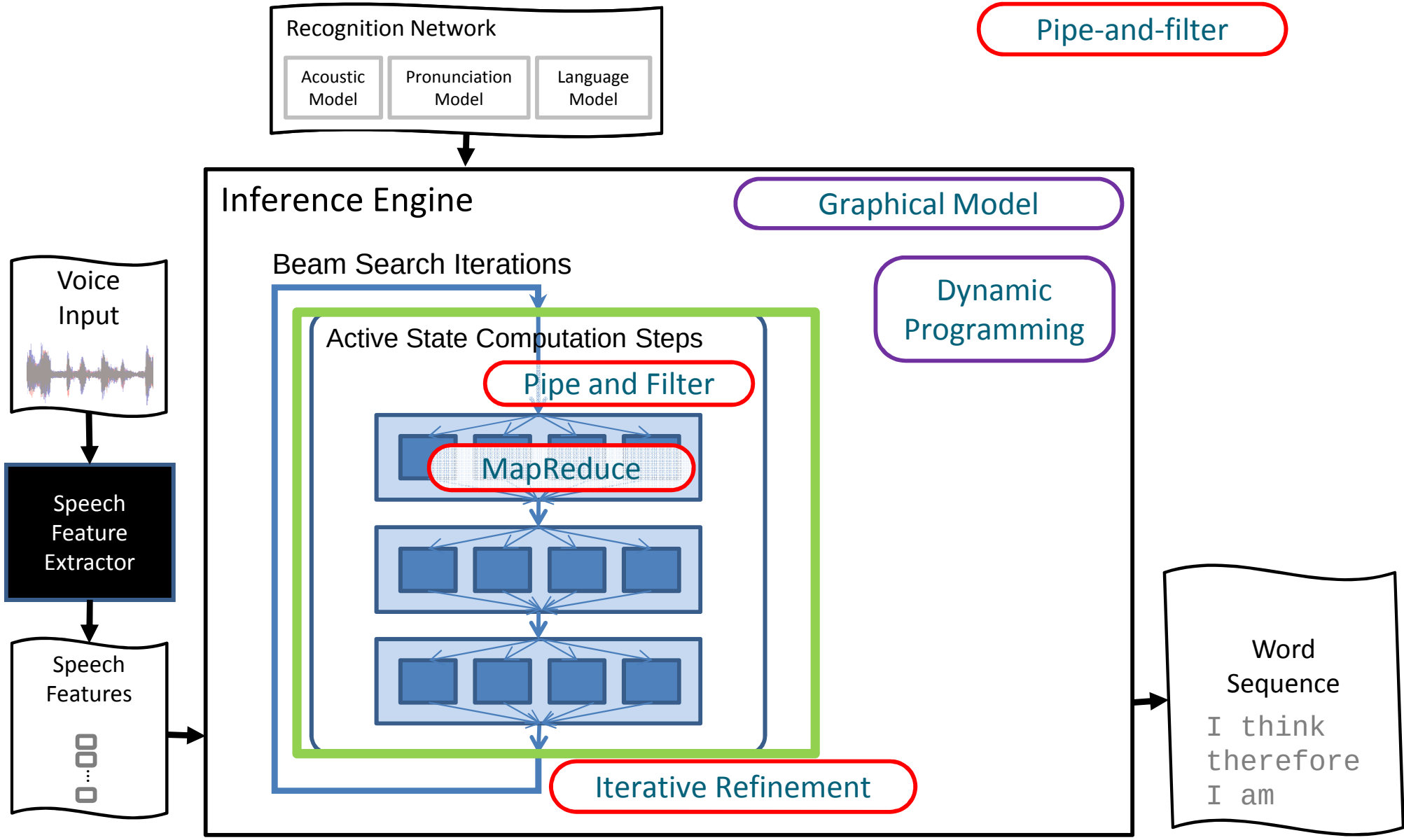
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Large vocabulary continuous speech recognition
- Summary

Large Vocabulary Continuous Speech Recognition



- Inference engine based system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan) [10,15,9]
- Modular and flexible setup
  - Shown to be effective for Arabic, English, Japanese, and Mandarin

LVCSR Software Architecture



Key computation: HMM Inference Algorithm

An instance of: Graphical Models

Implemented with: Dynamic Programming

- Finds the most-likely sequence of states that produced the observation

$$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

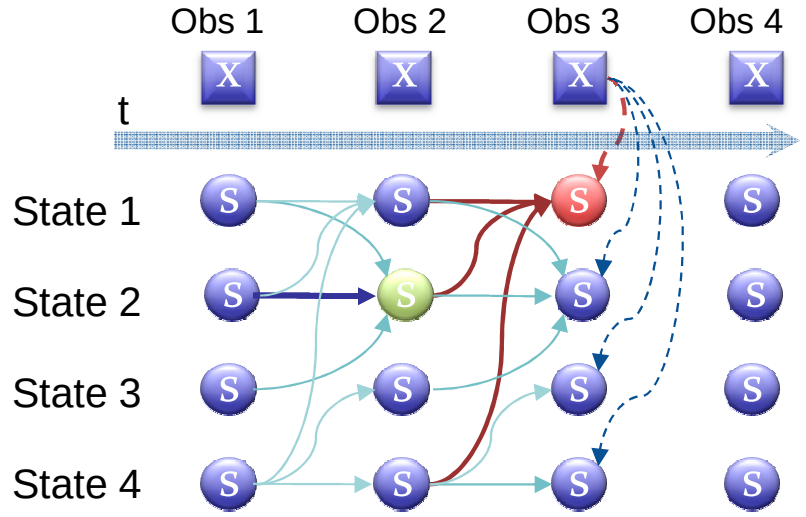
GMM

Frontier

Rec Network

Transition Probability

Viterbi Algorithm



Legends:

A State

An Observation

$P(x_t|s_t)$

$m\,[t\text{-}1][s_{t-1}]$

$P(s_t|s_{t-1})$

$m\,[t][s_t]$

Markov Condition:

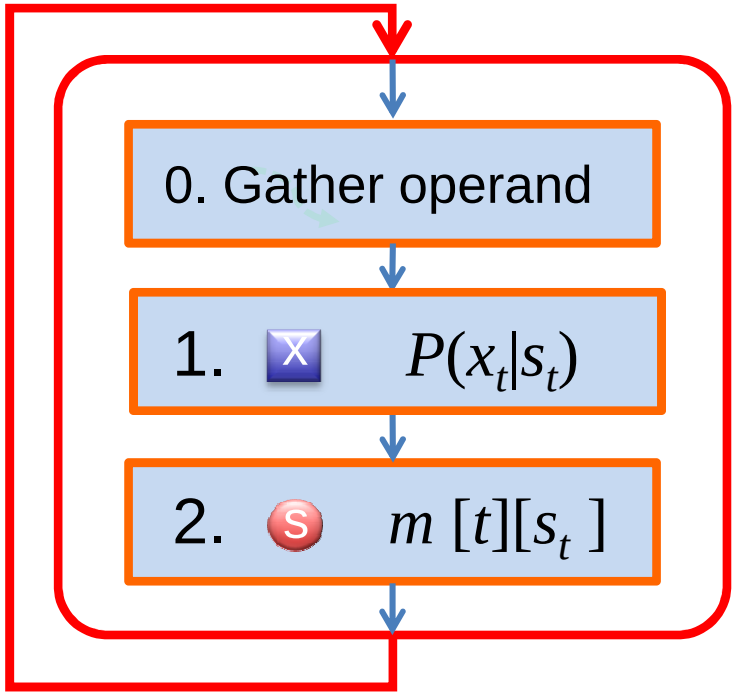
$$m[t][s_t] \doteq \max_{s_0,\ldots,s_{t-1}} P(x_0,\ldots,x_t,s_0,\ldots,s_{t-1},s_t)$$

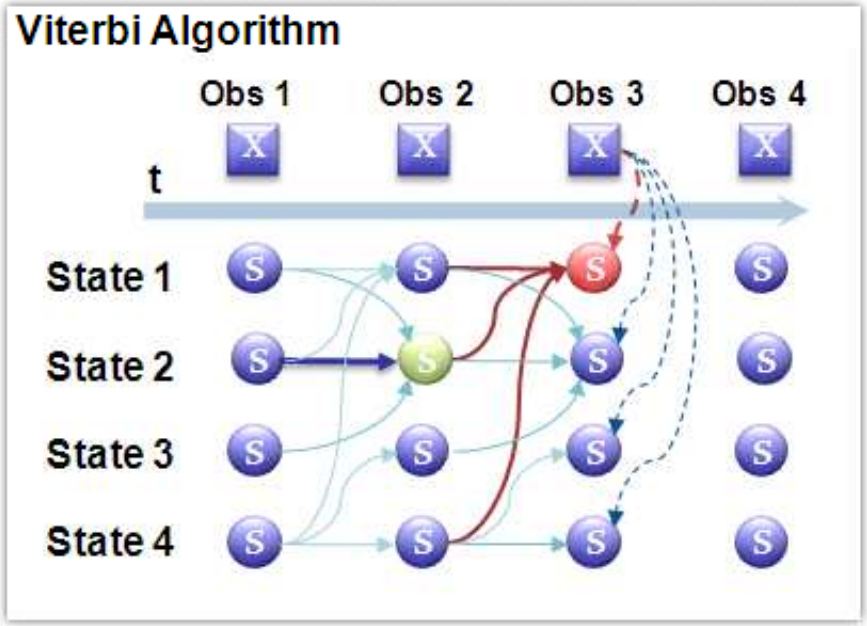
J. Chong, Y. Yi, A. Faria, N.R. Satish and K. Keutzer, "Data-Parallel Large Vocabulary Continuous Speech Recognition on Graphics Processors", Emerging Applications and Manycore Arch. 2008, pp. 23-35, June 2008

Inference Engine in LVCSR

- Three steps of inference
  0. Gather operands from irregular data structure to runtime buffer
  1. Perform observation probability computation
  2. Perform graph traversal computation

Parallelism in the inference engine:

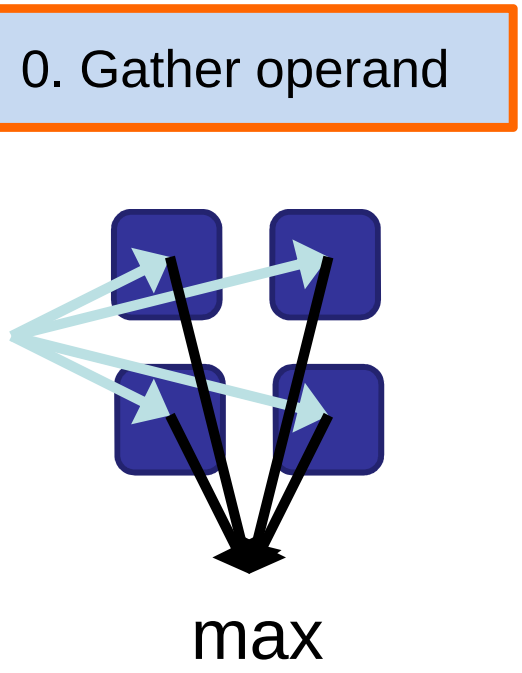




Each Filter is a Map Reduce
0. Gather operands

$$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

- Gather and coalesce each of the above operands for every s_t
- Facilitates opportunity for SIMD

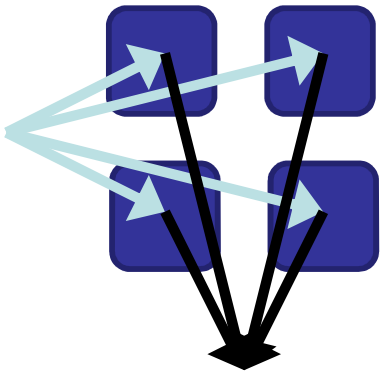


Each Filter is Map Reduce
1. observation probability computation

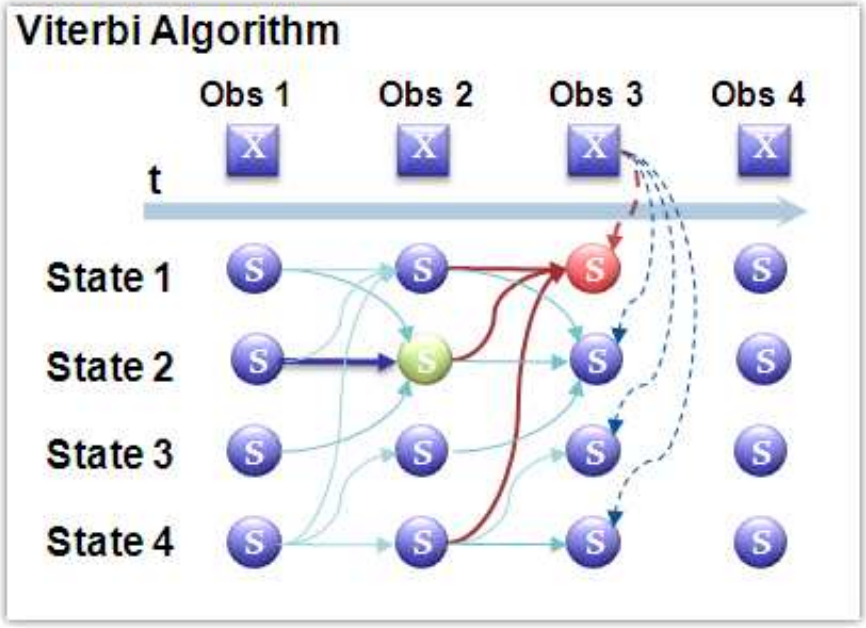
$$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

- Gaussian Mixture Model Probability
- Probability that given this feature-frame (e.g. 10ms) we are in this state/phone

1. X $P(x_t|s_t)$



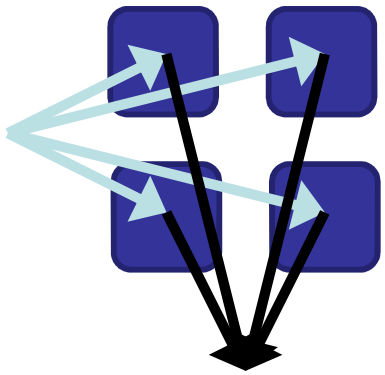
max



Each Filter is Map Reduce
2. graph traversal computation

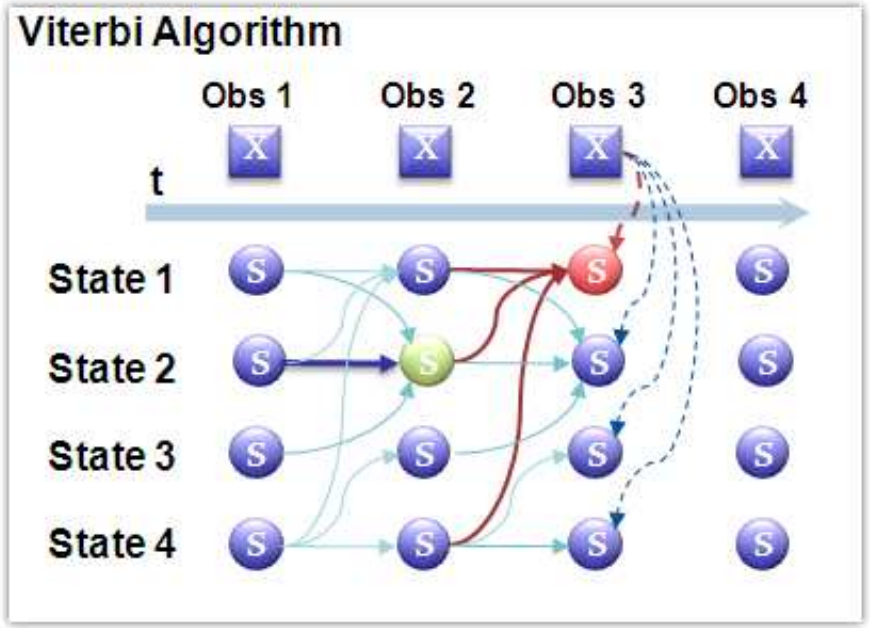
- **Map** probability computation across distributed data sets – perform multiplication as below
- **Reduce** the results to find the maximumly likely states

$$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

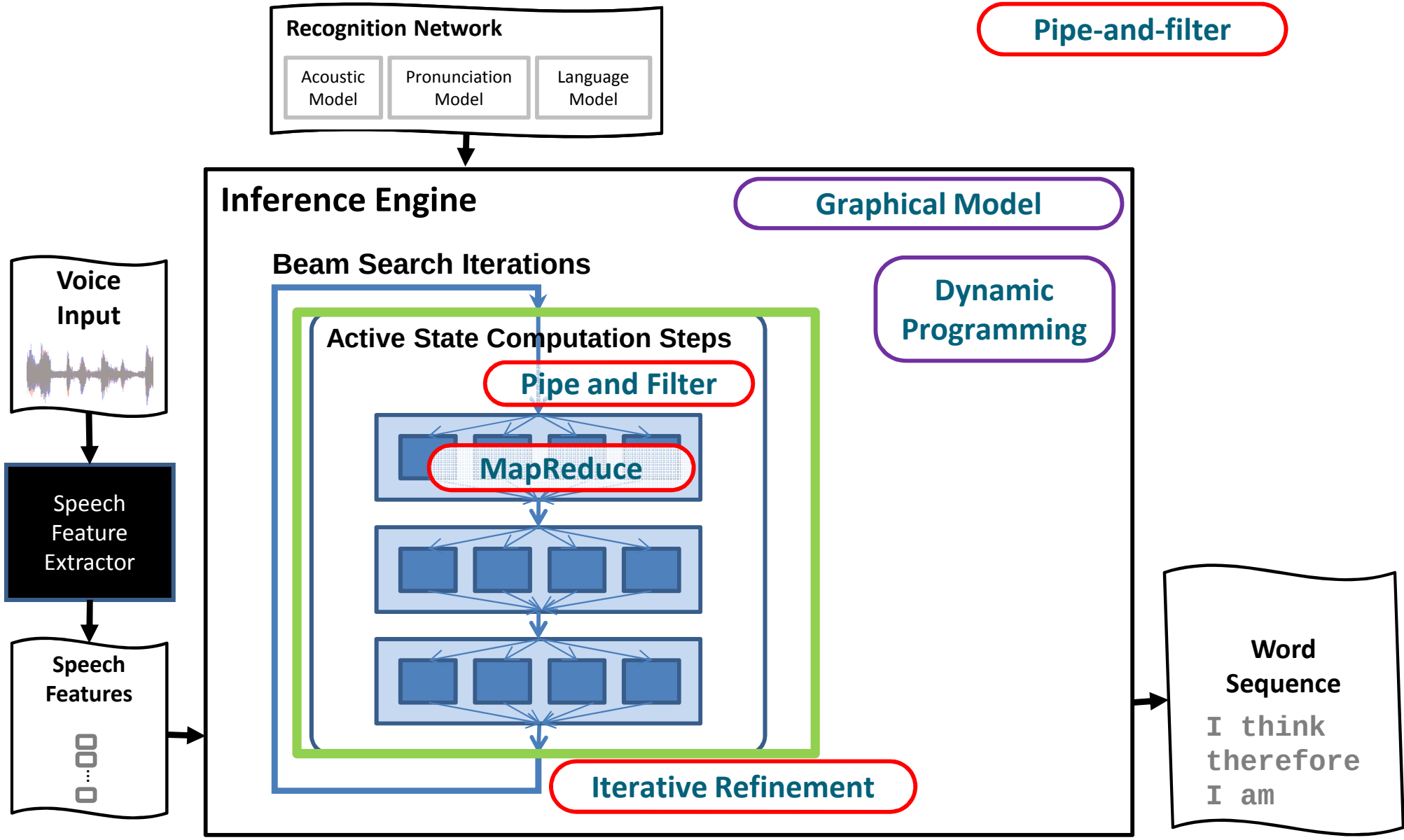


max

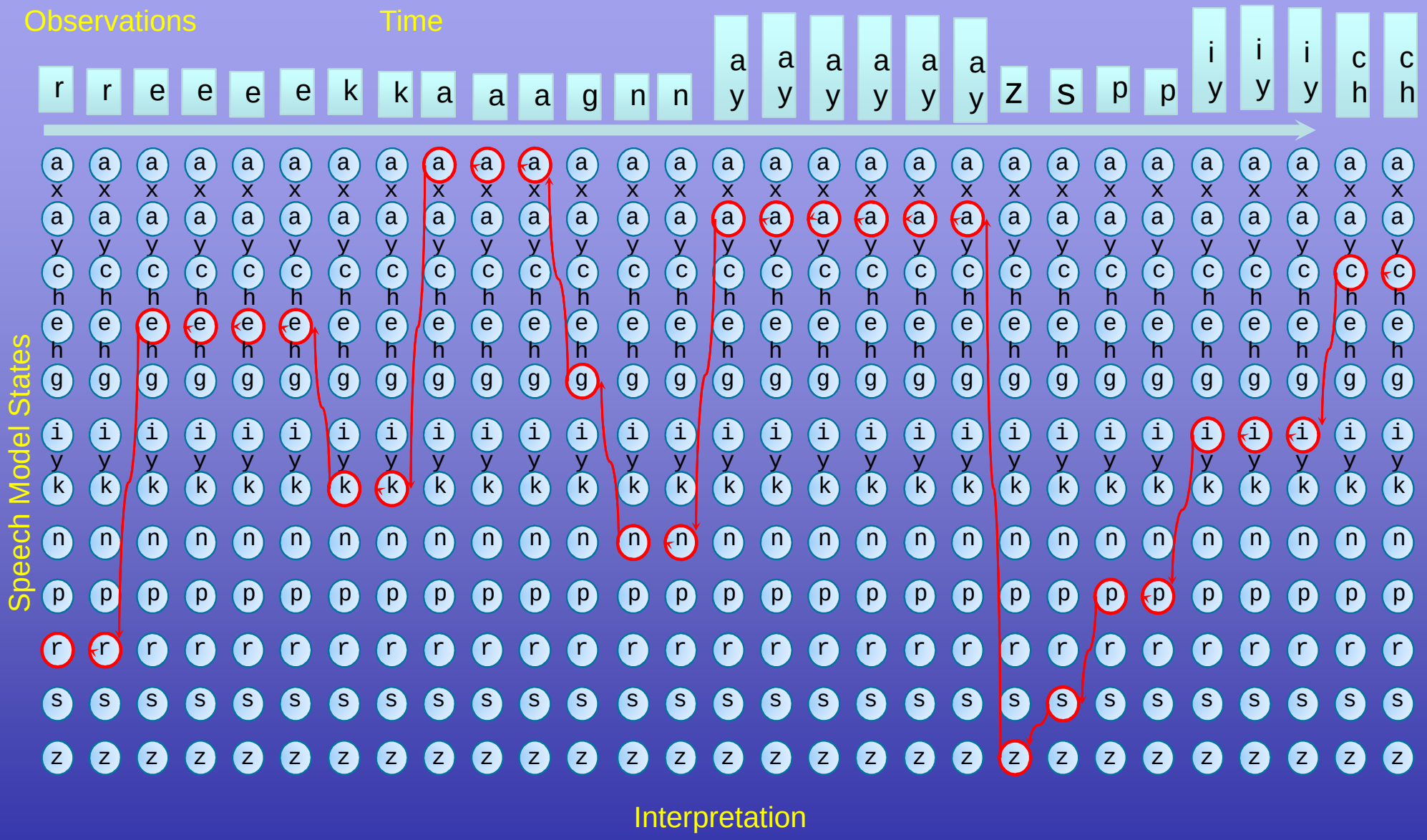
2. S $m[t][s_t]$



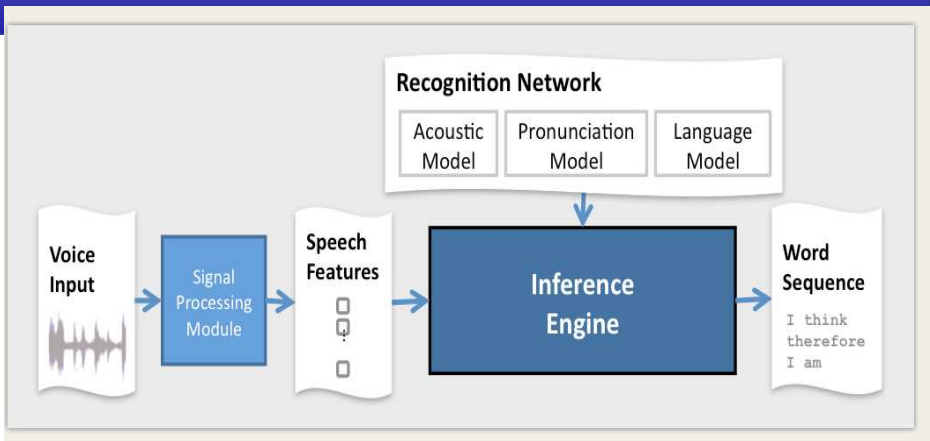
LVCSR Software Architecture



Speech Recognition with HMM



Speech Recognition Results

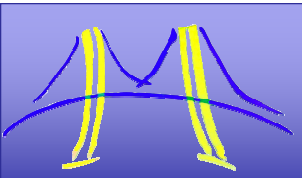


- Input: Speech audio waveform
- Output: Recognized word sequences

- Achieved 11x speedup over sequential version
- Allows 3.5x faster than real time recognition

- Our technique is being deployed in a hotline call-center data analytics company
- Used to search content, track service quality and provide early detection of service issues

Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010



Outline

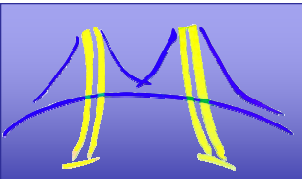
PARLAB

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Game development, as presented in:

→ **UC Berkeley undergraduate course CS194 Engineering Parallel Software, Fall 2010**

- Summary

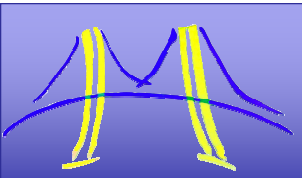
From research concept to full undergraduate class: **2 years**



Video Game Physics

PAMAS

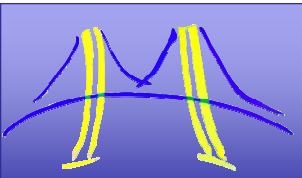
- Modern 3D Video Game Engines are simulations of a virtual world
  - Simulation of physicsis ***just realistic enough*** to be convincing. Physical accuracy (e.g. fidelity to CFD equations) is secondary to visual appearance and compute budget.
  - "Pre-Computed" physics == animations
- **Scenery**: E.G. Buildings, Trees, Terrain, ...
  - Can collide with non-Scenery objects, but don't move
  - Usually immutable -- appearance is pre-computable
- **Physical Objects**: E.G. vehicles, characters, projectiles, ...
  - Collide, explode, move: fully dynamic
  - Rigid-body simulations, Vehicle simultaion, "Ragdoll Corpses"
- **Non-Physical Objects**: E.G. lights, smoke plumes,
  - Most "Physical" computations in games are actually quick, cheap hacks that give the visual appearance of physical accuracy



Other sources of computation

PAMAS

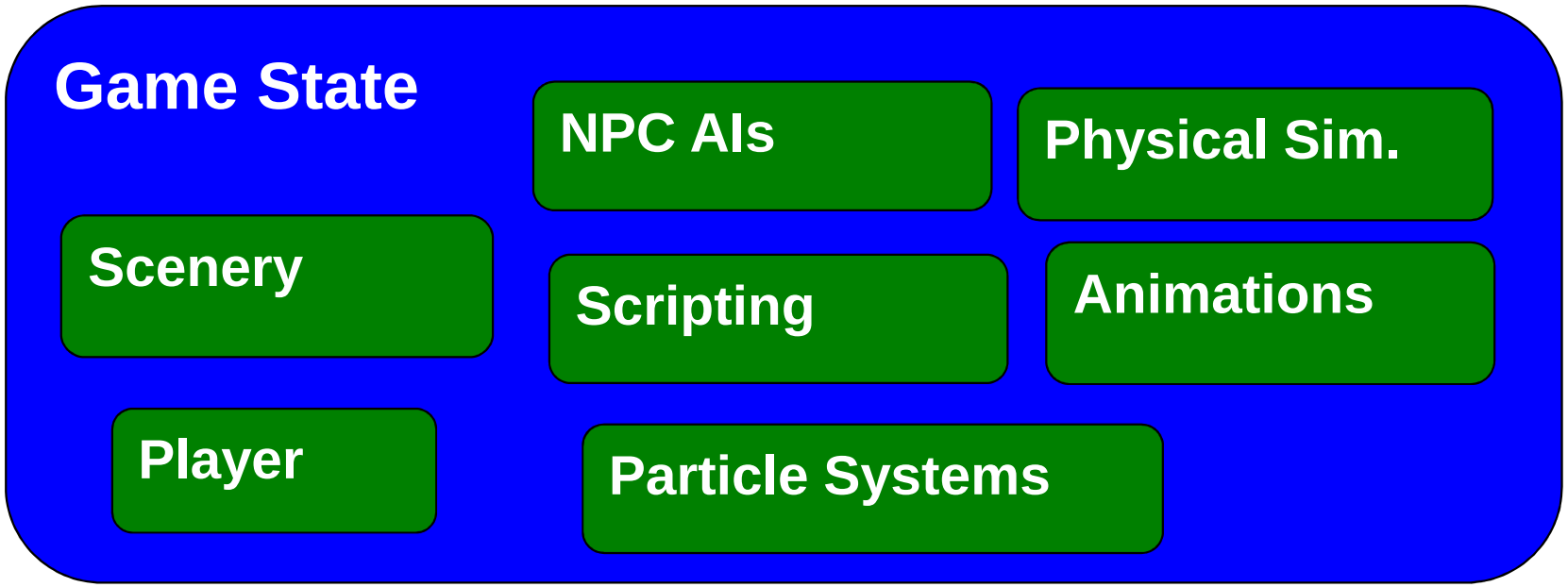
- **Animation:** Updating the visual state and physical configuration of characters and objects: crucial for realism, closely coupled with AI
- **"Artificial Intelligence" (AI):** Mostly scripts and ad-hoc decision trees, but physical queries (am I close to something interesting?) and graph-searching / path planning
- **Damage and Destruction**: E.g. vehicles, buildings, characters can be destroyed, burned, modified. I.E. produce physical objects!
- **Audio:** The many sources of sound in a game's scene must be processed and combined according to occlusion, distance from player, etc.
- **Input System:** Currently, very trivial ... but with the introduction of accelerometers, video cameras, etc. could get very interesting.

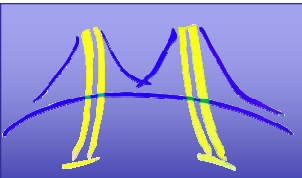


Architecting A Video Game

PAMAS

- The Game Engine's purpose is to maintain, update, and display the state of the Virtual World it is simulating
  - Each stateful sub-system
- How can we organize the structure of this software?

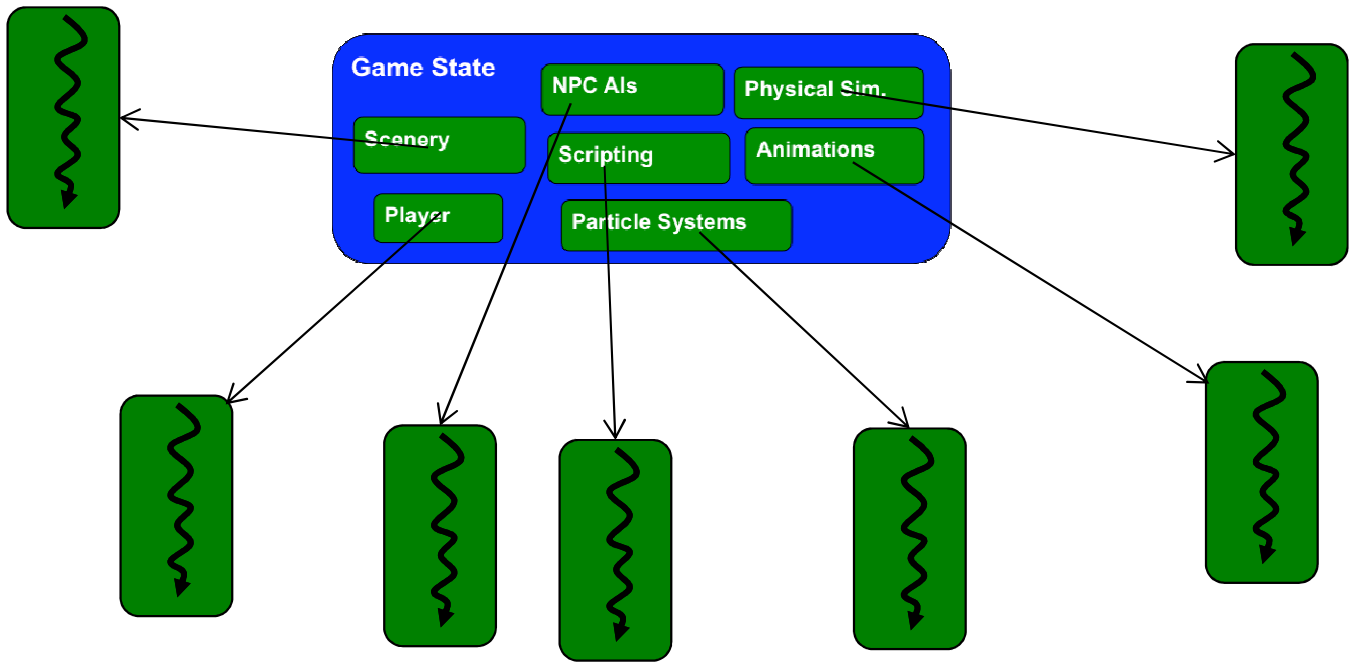




Anti-Solution: Coarse Multithreading

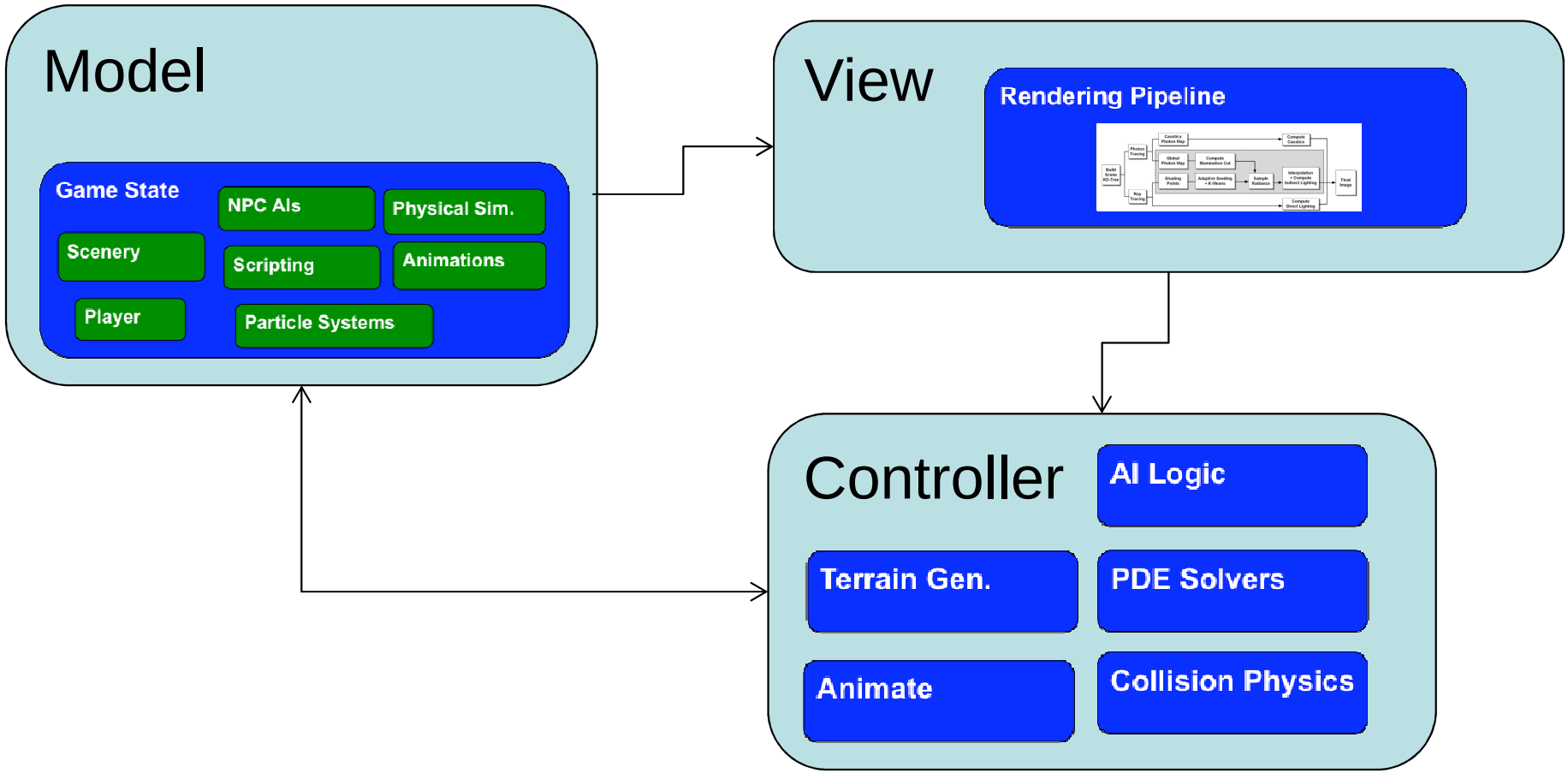
PAMAS

- Assign each sub-module to a different CPU Thread
  - Was the first pass approach of Game Engine implementers
- Subsystem interactions become Mutual Exclusions on Shared Data
- In Testing, Verification and Debugging, you are at the mercy of the OS's non-deterministic, non-repeatable thread scheduler
- Load Balancing: the "Physical Simulation" thread will have much more work than the "Scripting" thread



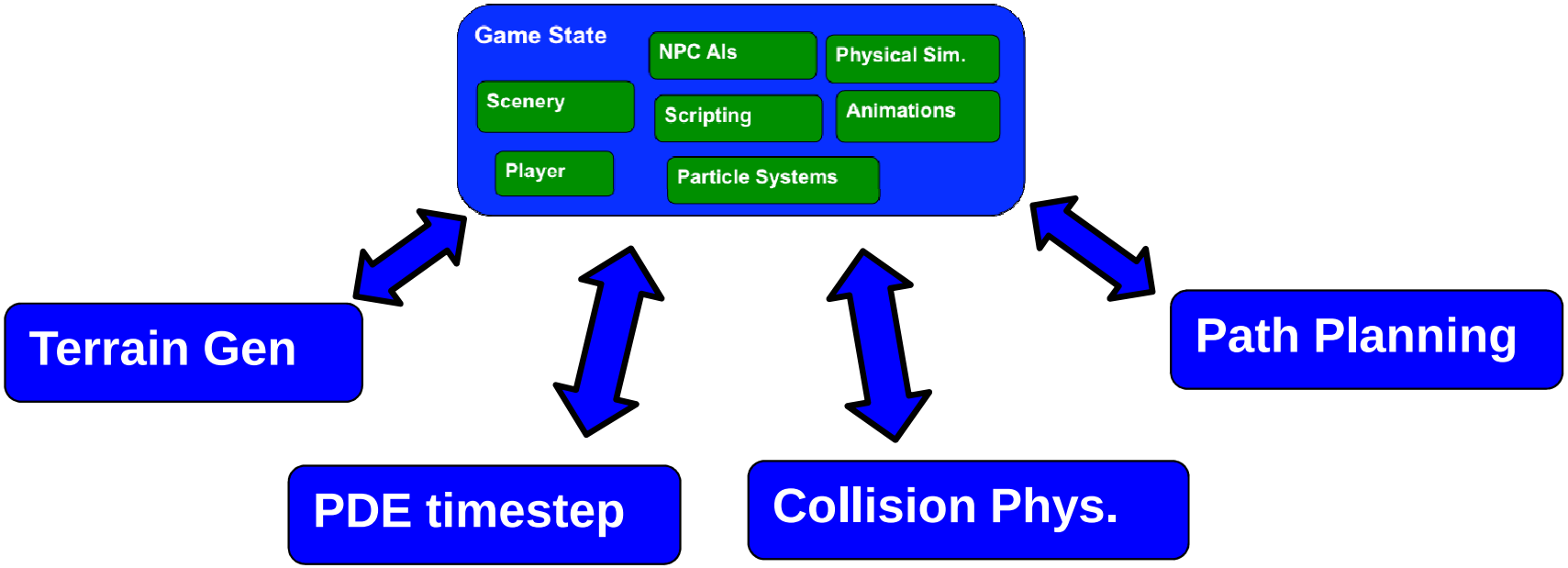
Architecture, one choice: MVC Pattern

- Model View Controller Structural Pattern:
  - Model: maintain the system's state, update "observers"
  - View: Renders the model for interaction with User
  - Controller: Initiates responses to Input by modifying the Model



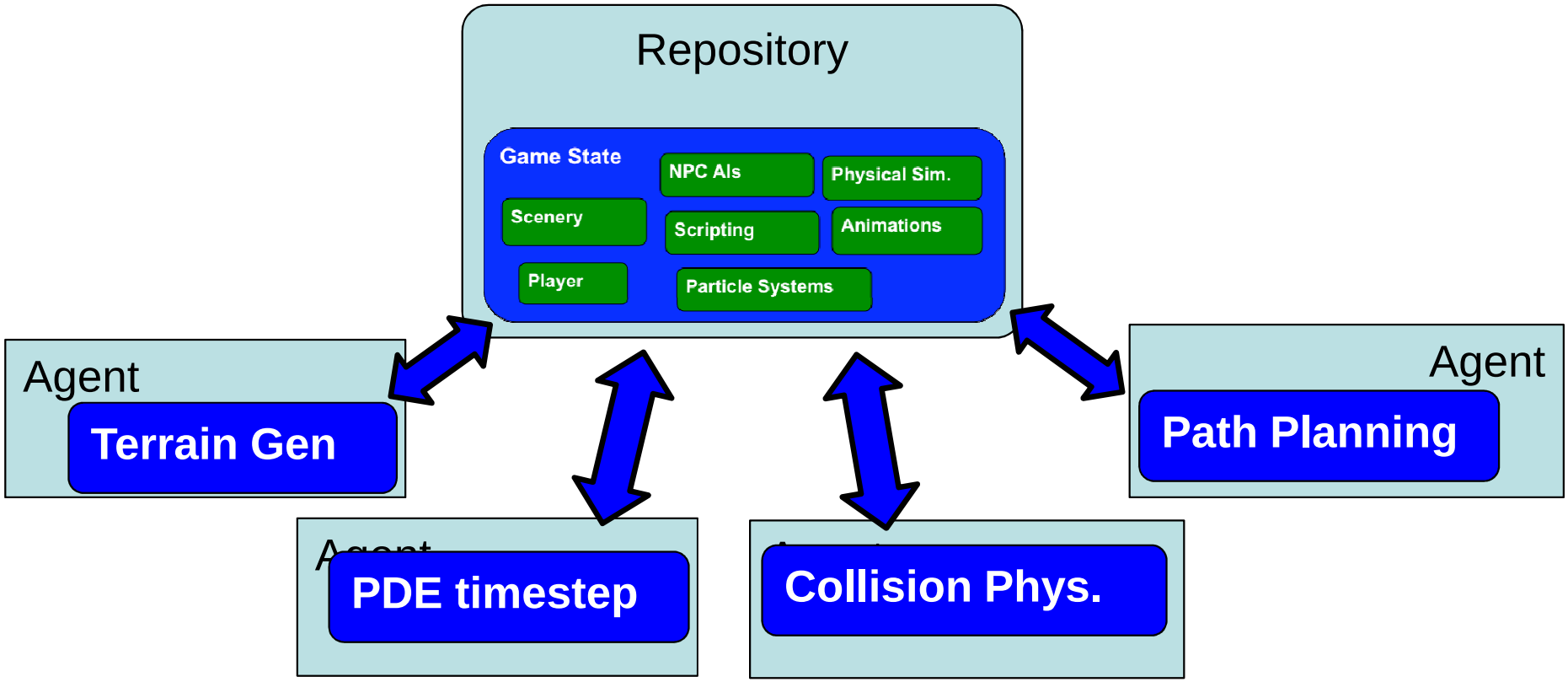
Module-Level Parallelism

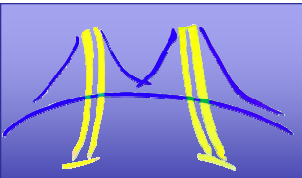
- Executing multiple sub-modules in parallel present problems:
  - Modules share data: some have read-only access to state that other modules will update
- We need a way of managing access to the Data. Suggestions?



Agent and Repository

- Agent and Repository pattern is an approach to assigning modules responsibility in accessing a shared data structure
  - Repository is responsible for partitioning of the data and scheduling of the agents
  - Agents execute when Repository Manager allows them to

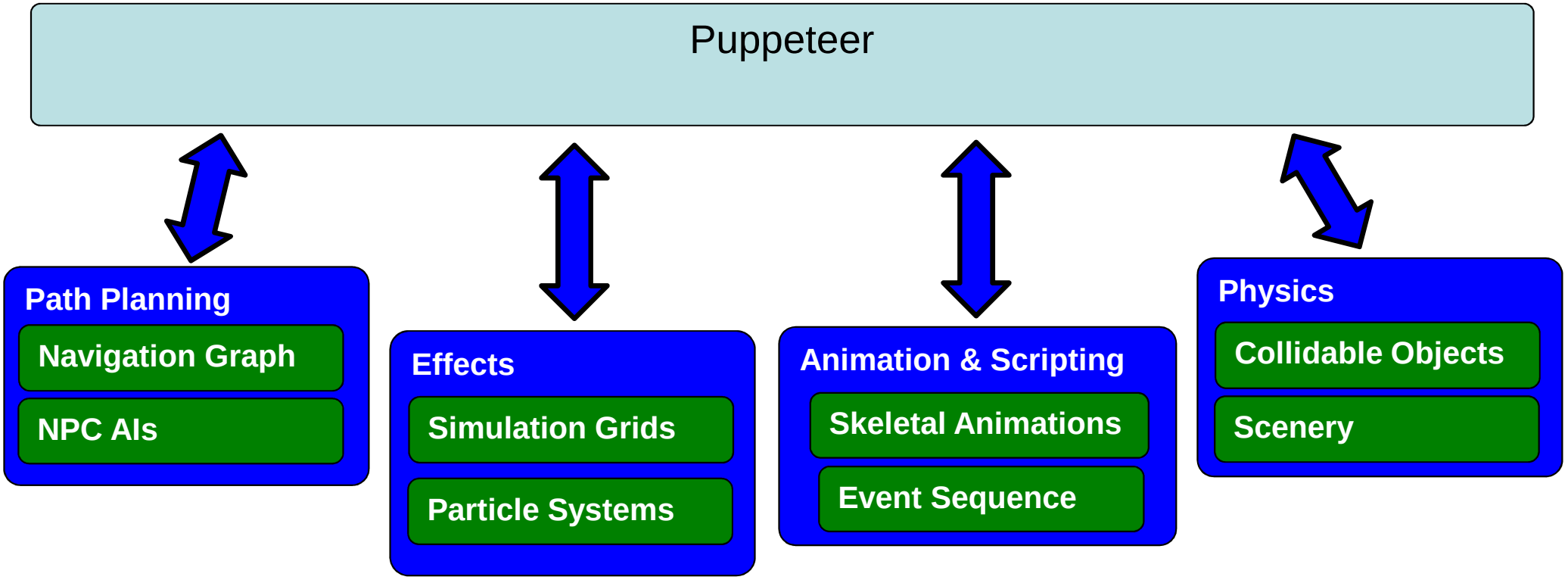


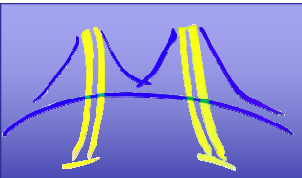


Puppeteer Pattern

PAMAS

- An alternative: The Puppeteer. Sub-modules are responsible for the management of their own data (no central repository)
  - Repository Manager owned data in A&R
  - Puppeteer only owns the channels of communication
- Communication protocol: e.g. Subscribe & Notify
  - Or "Baked" into the puppeteer implementation





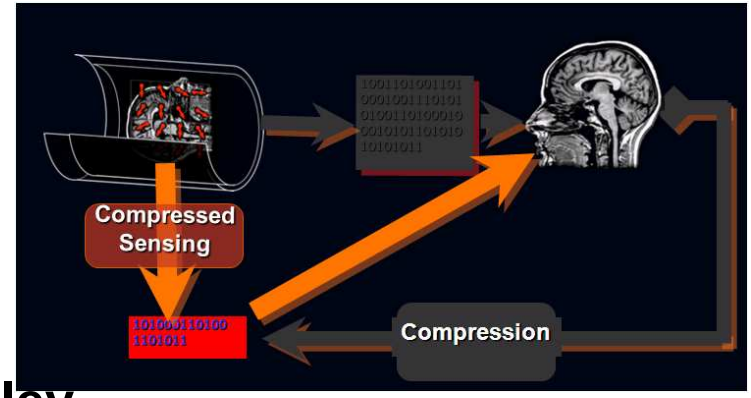
Outline

PARLAB

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Pediatric MRI
- Summary

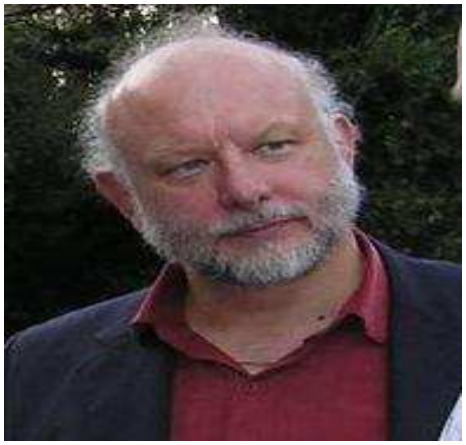
Fast MRI Reconstruction

Miki Lustig
EECS, UC Berkeley



Prof. Shreyas Vasanawala, MD, PhD
Stanford Medical School

Mark Murphy



Kurt Keutzer

Compelling Application: Fast, Robust Pediatric MRI

Pediatric MRI is difficult:

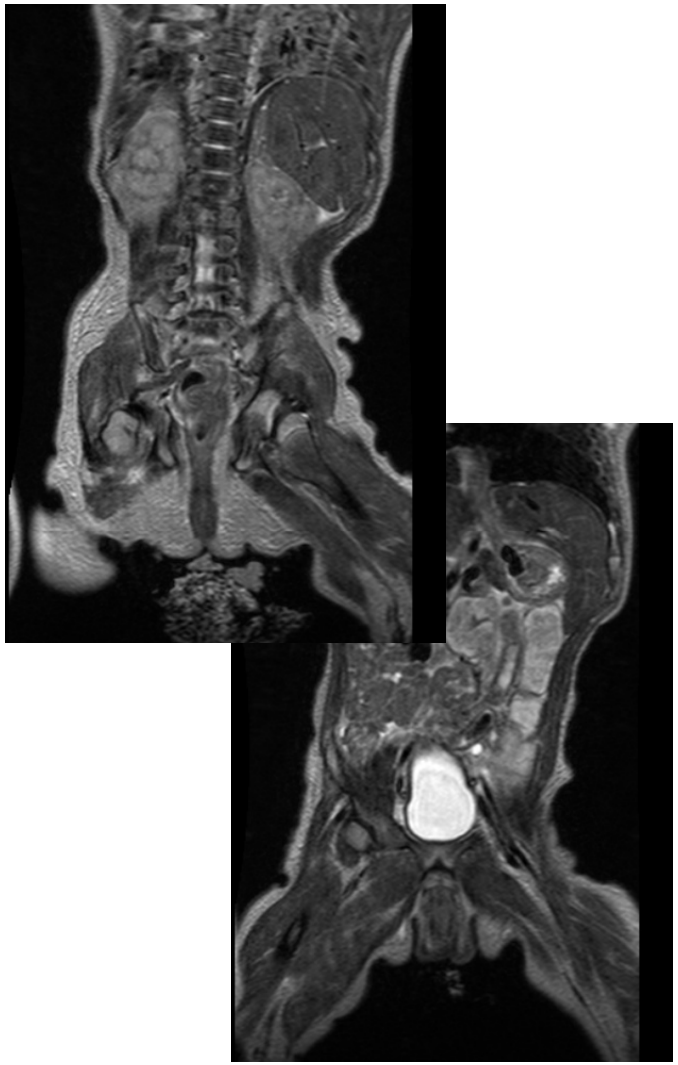
- Children cannot sit still, breathhold
- Low tolerance for long exams
- Anesthesia is costly and risky

Like to accelerate MRI acquisition

- Advanced MRI techniques exist, but require data- and compute- intense algorithms for image reconstruction

Reconstruction must be fast, or time saved in accelerated acquisition is lost in computing reconstruction

- Slow reconstruction times are a non-starter for clinical use

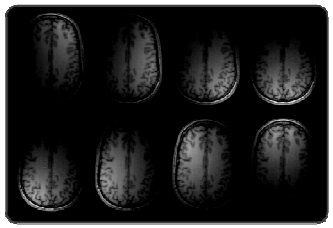
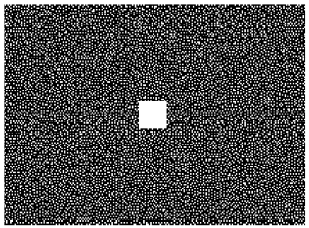
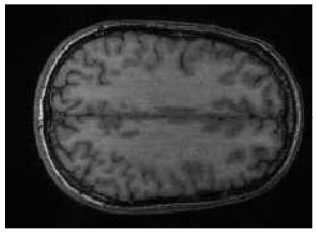


Domain Experts and State-of-the-Art Algorithms

Collaboration with MRI Researchers:

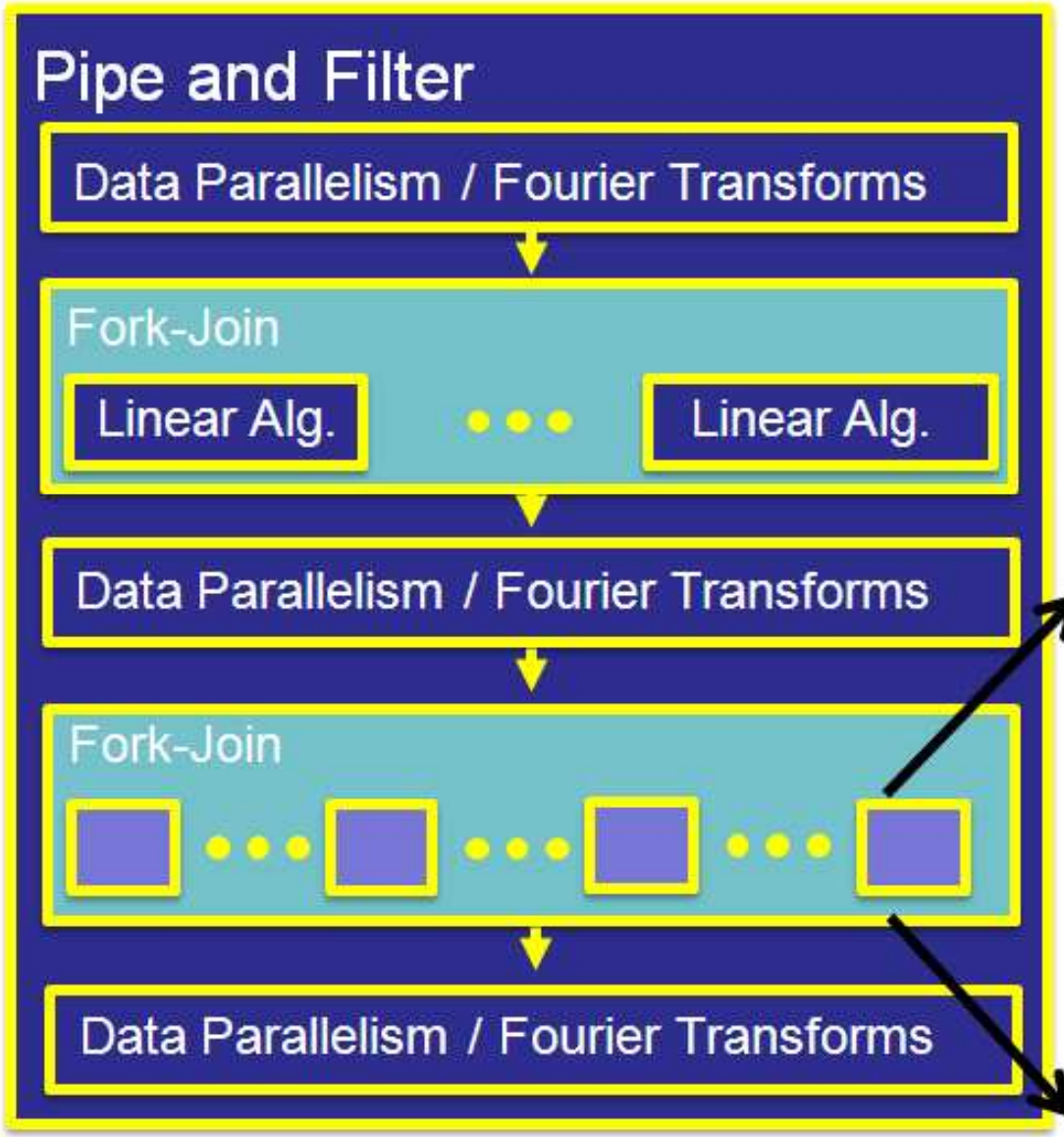
- Miki Lustig, Ph.D., Berkeley EECS
- Marc Alley, Ph.D., Stanford EE
- Shreyas Vasanawala, M.D./Ph.D., Stanford Radiology

Advanced MRI: Parallel Imaging and Compressed Sensing to dramatically reduce MRI image acquisition time

$$\begin{aligned} \text{minimize} \quad & ||Wx||_1 \\ \text{s.t} \quad & \mathbf{F}_\Omega x = y, \\ & ||\mathbf{G}x - x||_2 < \varepsilon \end{aligned}$$

SW architecture of image reconstruction



Iterative POCS Algorithm:

1. Apply SPIRiT Operator:
$$x_c \leftarrow \sum_j g_{cj} * x_j$$
2. Wavelet Soft-Thresholding
$$x \leftarrow W S_\lambda \{W^* x\}$$
3. Fourier-space projection
$$x \leftarrow F(P^T y + P_c^T P_c F^* x)$$

Iter. Refinement / Spectral Method

- Data Parallelism / Convolutions
- Data Parallelism / Wavelet xforms
- Data Parallelism / Fourier xforms

Game-Changing Speedup

100X faster reconstruction

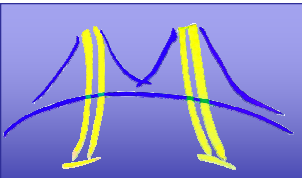
Higher-quality, faster MRI

This image: 8 month-old patient with cancerous mass in liver

- 256 x 84 x 154 x 8 data size
- Serial Recon: 1 hour
- Parallel Recon: 1 minute

Fast enough for clinical use

- Software currently deployed at Lucile Packard Children's Hospital for clinical study of the reconstruction technique



Summary

PAMAS

What's the point?

- Parallel design patterns give a new powerful viewpoint toprogrammers

- Enable us to disentangle the big fuzzy ball of yarn of computation
  - add 20 IQ points to our problem solving (as per Alan Kay)

- Our Pattern language helps you to write good parallel code

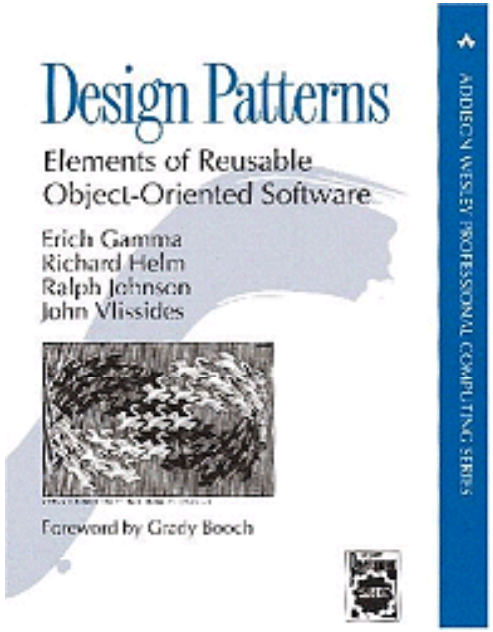
Software Design Patterns in education: lessons from history?

Early days OO

Perception: Object oriented? Isn't that just an academic thing?

Usage: specialists only. Mainstream regards with indifference or anxiety.

Performance: not so good.



1994

Now

Perception: OO=programming

Isn't this how it was always done?

Usage: cosmetically widespread, some key concepts actually deployed.

Performance: so-so, masked by CPU advances until now.

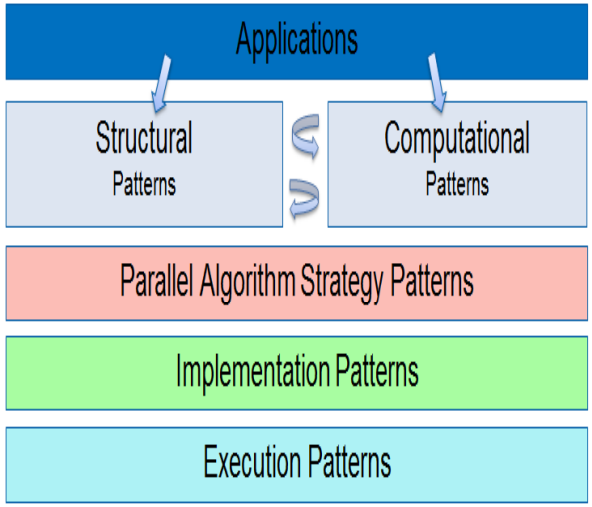
Software Design Patterns in education: lessons from history?

Now

Perception: Parallel programming? Isn't that just an HPC thing?

Usage: specialists only. Mainstream regards with indifference or anxiety.

Performance: very good, for the specialists.



2012

Future

Perception: PP=programming

Isn't this how it was always done?

Usage: widespread, key concepts actually deployed.

Performance: broadly sufficient. Application domains greatly expanded.