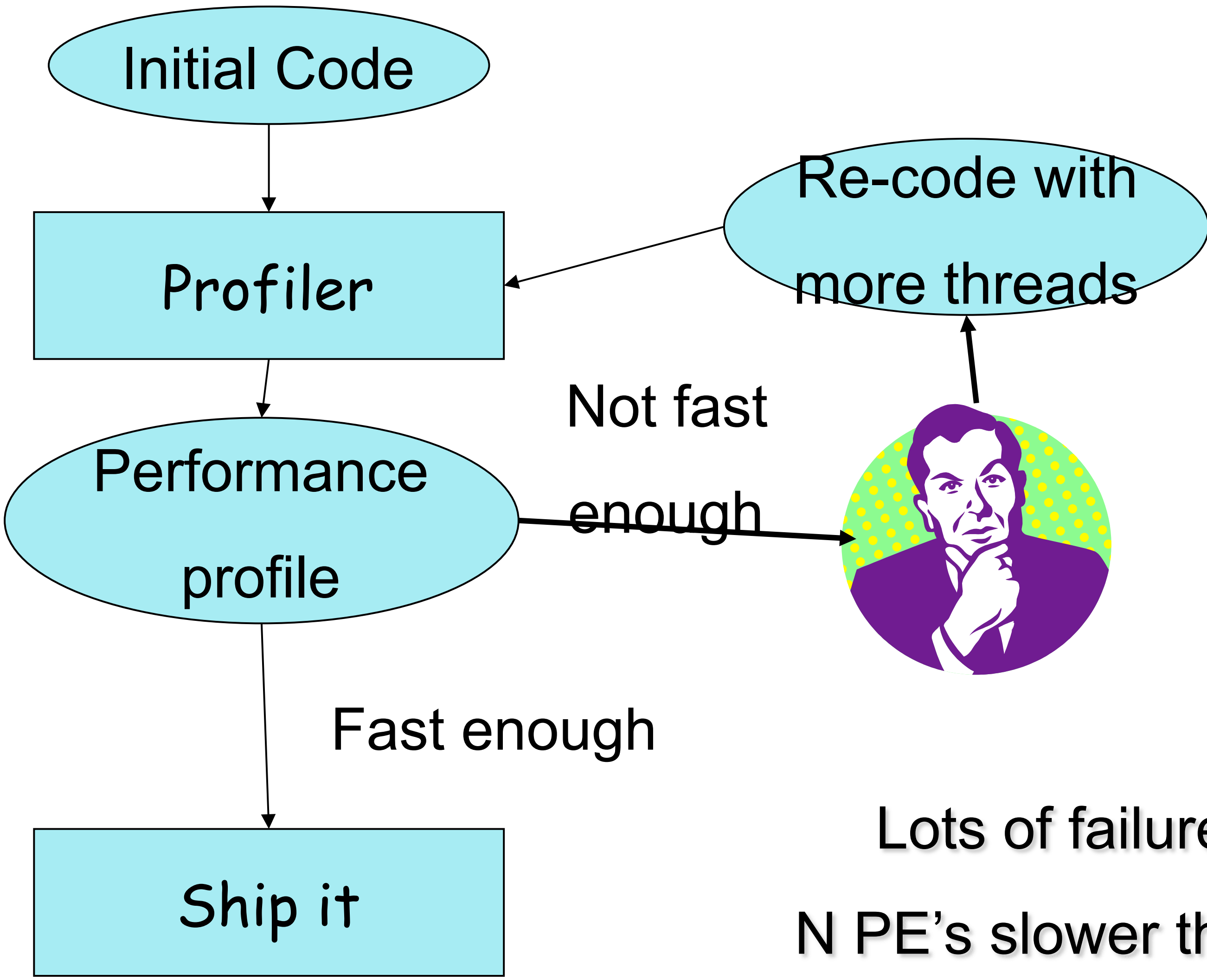
PARLab Parallel Boot Camp

Architecting Parallel Software with Patterns

Kurt Keutzer
Electrical Engineering and Computer Sciences
University of California, Berkeley

Assumption #1: How **not** to develop parallel code



Lots of failures

N PE's slower than 1

Steiner Tree Construction Time By Routing Each Net in Parallel

Benchmark	Serial	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
adaptec1	1.68	1.68	1.70	1.69	1.69	1.69
newblue1	1.80	1.80	1.81	1.81	1.81	1.82
newblue2	2.60	2.60	2.62	2.62	2.62	2.61
adaptec2	1.87	1.86	1.87	1.88	1.88	1.88
adaptec3	3.32	3.33	3.34	3.34	3.34	3.34
adaptec4	3.20	3.20	3.21	3.21	3.21	3.21
adaptec5	4.91	4.90	4.92	4.92	4.92	4.92
newblue3	2.54	2.55	2.55	2.55	2.55	2.55
average	1.00	1.0011	1.0044	1.0049	1.0046	1.0046

Assumption #2: This won't help either

Code in new cool language

Profiler

Re-code with cool language

Not fast enough

Performance profile

Fast enough

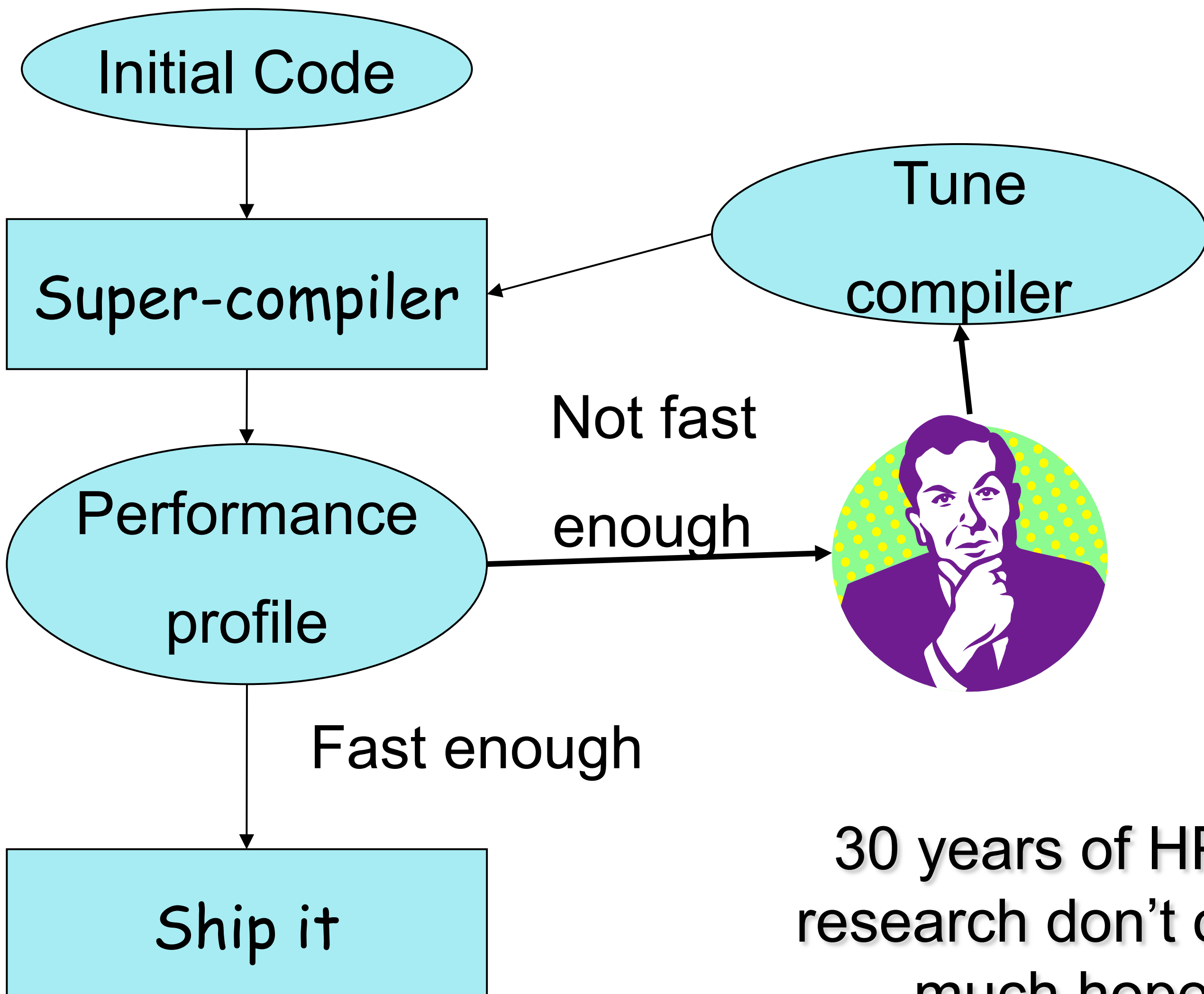
Ship it

After 200 parallel languages where's the light at the end of the tunnel?

Parallel Programming environments in the 90's

ABCPL	CORRELATE	GLU	Mentat	Parafrase2	pC++
ACE	CPS	GUARD	Legion	Paralation	SCHEDULE
ACT++	CRL	HAsL.	Meta Chaos	Parallel-C++	SciTL
Active messages	CSP	Haskell	Midway	Parallaxis	POET
Adl	Cthreads	HPC++	Millipede	ParC	SDDA.
Adsmith	CUMULVS	JAVAR.	CparPar	ParLib++	SHMEM
ADDAP	DAGGER	HORUS	Mirage	ParLin	SIMPLE
AFAPI	DAPPLE	HPC	MpC	Parmacs	Sina
ALWAN	Data Parallel C	IMPACT	MOSIX	Parti	SISAL.
AM	DC++	ISIS.	Modula-P	pC	distributed smalltalk
AMDC	DCE++	JAVAR.	Modula-2*	pC++	SMI.
AppLeS	DDD	JADE	Multipol	PCN	SONiC
Amoeba	DICE.	Java RMI	MPI	PCP:	Split-C.
ARTS	DIPC	javaPG	MPC++	PH	SR
Athapascan-0b	DOLIB	JavaSpace	Munin	PEACE	Sthreads
Aurora	DOME	JIDL	Nano-Threads	PCU	Strand.
Automap	DOSMOS.	Joyce	NESL	PET	SUIF.
bb_threads	DRL	Khoros	NetClasses++	PETSc	Synergy
Blaze	DSM-Threads	Karma	Nexus	PENNY	Telegrphos
BSP	Ease .	KOAN/Fortran-S	Nimrod	Phosphorus	SuperPascal
BlockComm	ECO	LAM	NOW	POET.	TCGMSG.
C*.	Eiffel	Lilac	Objective Linda	Polaris	Threads.h++.
"C* in C	Eilean	Linda	Occam	POOMA	TreadMarks
C**	Emerald	JADA	Omega	POOL-T	TRAPPER
CarlOS	EPL	WWWinda	OpenMP	PRESTO	uC++
Cashmere	Excalibur	ISETL-Linda	Orca	P-RIO	UNITY
C4	Express	ParLin	OOF90	Prospero	UC
CC++	Falcon	Eilean	P++	Proteus	V
Chu	Filaments	P4-Linda	P3L	QPC++	ViC*
Charlotte	FM	Glenda	p4-Linda	PVM	Visifold V-NUS
Charm	FLASH	POSYBL	Pablo	PSI	VPE
Charm++	The FORCE	Objective-Linda	PADE	PSDM	Win32 threads
Cid	Fork	LiPS	PADRE	Quake	WinPar
Cilk	Fortran-M	Locust	Panda	Quark	WWWinda
CM-Fortran	FX	Lparx	Papers	Quick Threads	XENOOPS
Converse	GA	Lucid	AFAPI.	Sage++	XPC
Code	GAMMA	Maisie	Para++	SCANDAL	Zounds
COOL	Glenda	Manifold	Paradigm	SAM	ZPL

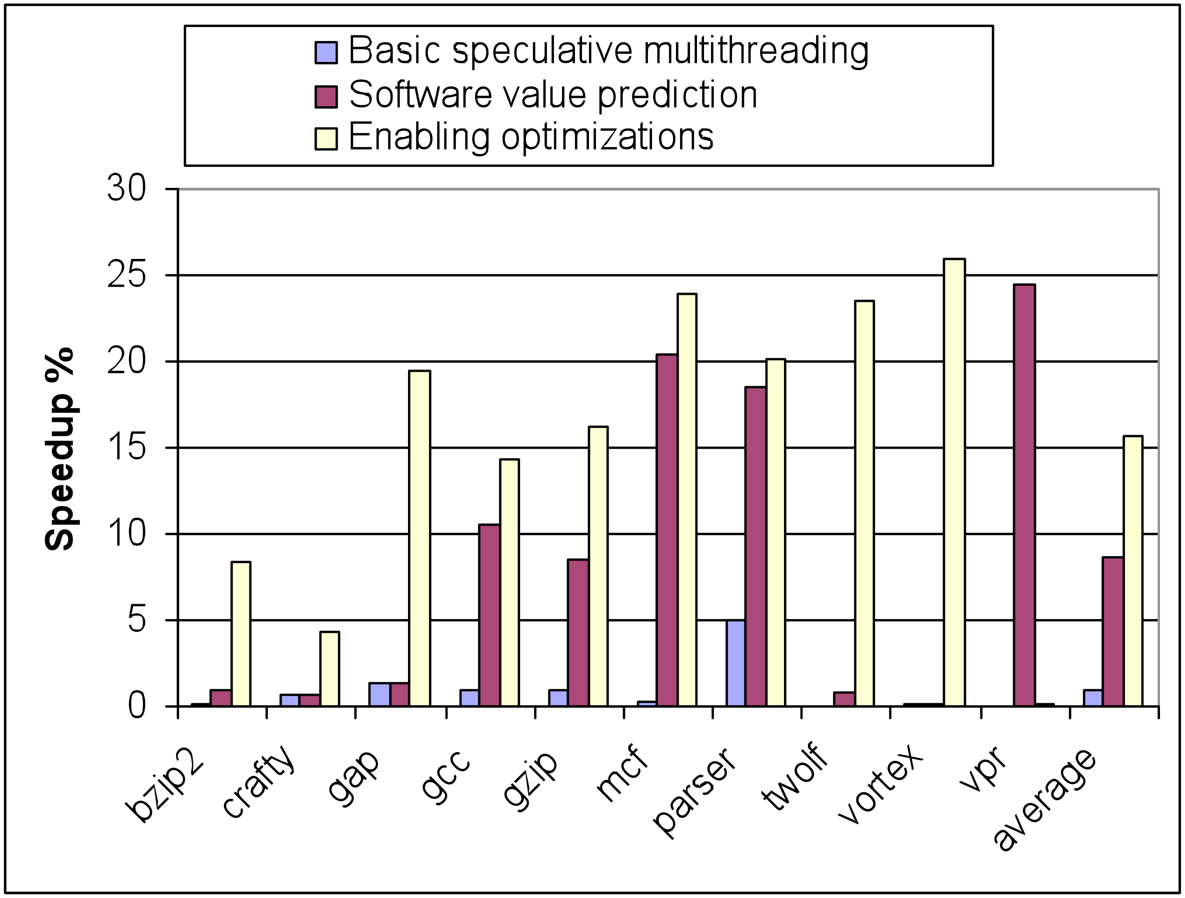
Assumption #3: Nor this



30 years of HPC research don't offer much hope

6

Automatic parallelization?



- Aggressive techniques such as speculative multithreading help, but they are not enough.
- Ave SPECint speedup of 8% … will climb to ave. of 15% once their system is fully enabled.
- There are no indications auto par. will radically improve any time soon.
- Hence, I do not believe Auto-par will solve our problems.

Results for a simulated dual core platform configured as a main core and a core for speculative execution.

Outline

- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns

Key Elements of Kurt's SW Education

- AT&T Bell Laboratories: CAD researcher and programmer
  - Algorithms: D. Johnson, R. Tarjan
  - Programming Pearls: S C Johnson, K. Thompson, (Jon Bentley)
  - Developed useful software tools:
    - » Plaid: programmable logic aid: used for developing 100's of FPGA-based HW systems
    - » CONES/DAGON: used for designing >30 application-specific integrated circuits
- Synopsys: researcher → CTO (25 products, ~15 million lines of code, $750M annual revenue, top 20 SW companies)
  - Super programming: J-C Madre, Richard Rudell, Steve Tjiang
  - Software architecture: Randy Allen, Albert Wang
  - High-level Invariants: Randy Allen, Albert Wang
- Berkeley: teaching software engineering and Par Lab
  - Took the time to reflect on what I had learned:
  - Architectural styles: Garlan and Shaw
    - » Design patterns: Gamma et al (aka Gang of Four), Mattson's PLPP
    - » A Pattern Language: Alexander, Mattson
    - » Dwarfs: Par Lab Team

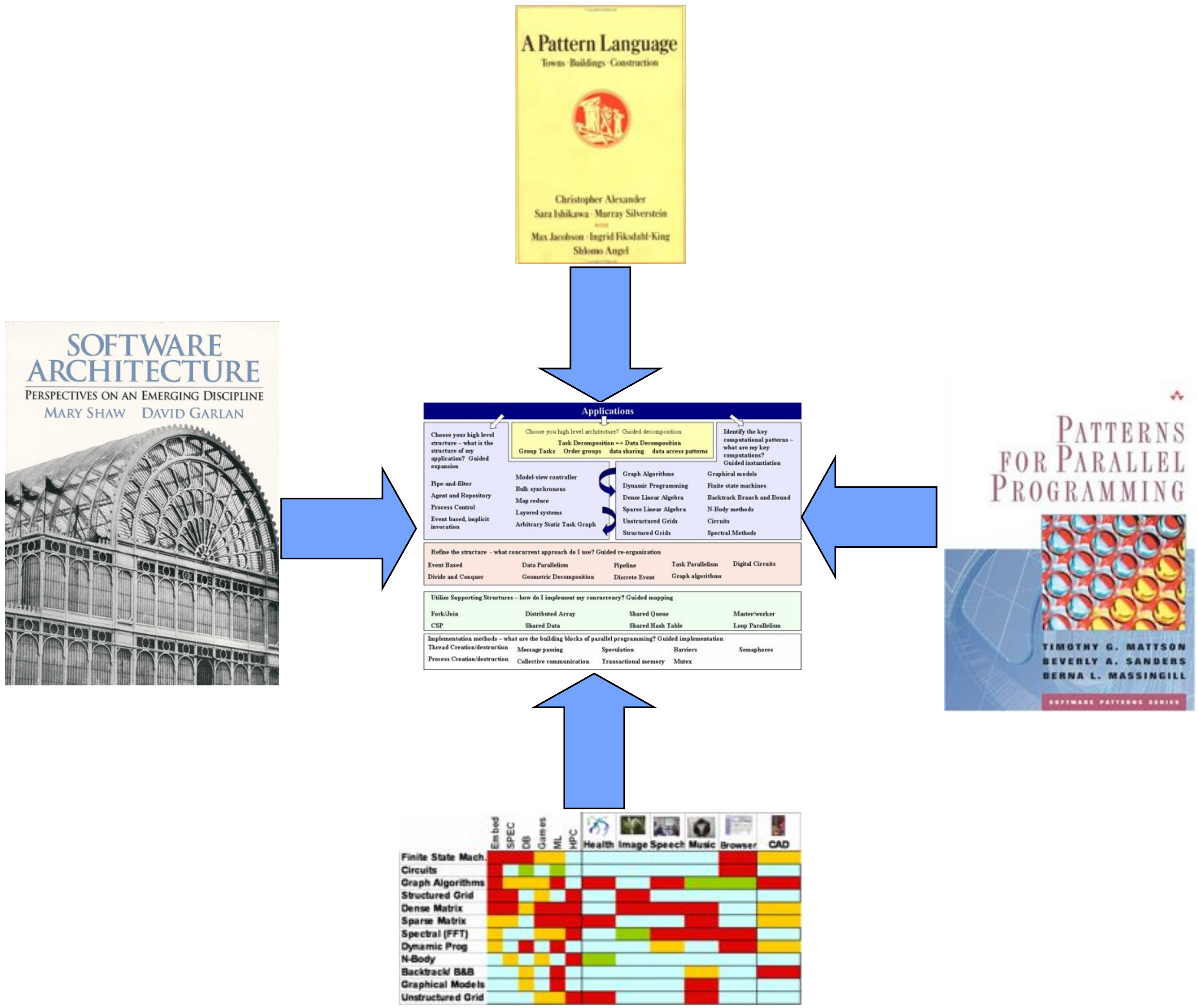
What I learned (the hard way)

- Software must be architected to achieve productivity, efficiency, and correctness
- SW architecture >> programming environments
  - >> programming languages
  - >> compilers and debuggers
  - (>>hardware architecture)
- Discussions with superprogrammers taught me:
  - Give me the right program structure/architecture I can use any programming language
  - Give me the wrong architecture and I'll never get there
- What I've learned when I had to teach this stuff at Berkeley:
- Key to architecture (software or otherwise) is design patterns and a pattern language
- Resulting software design then uses a hierarchy of software frameworks for implementation
  - Application frameworks for application (e.g. CAD) developers
  - Programming frameworks for those who build the application frameworks

Outline

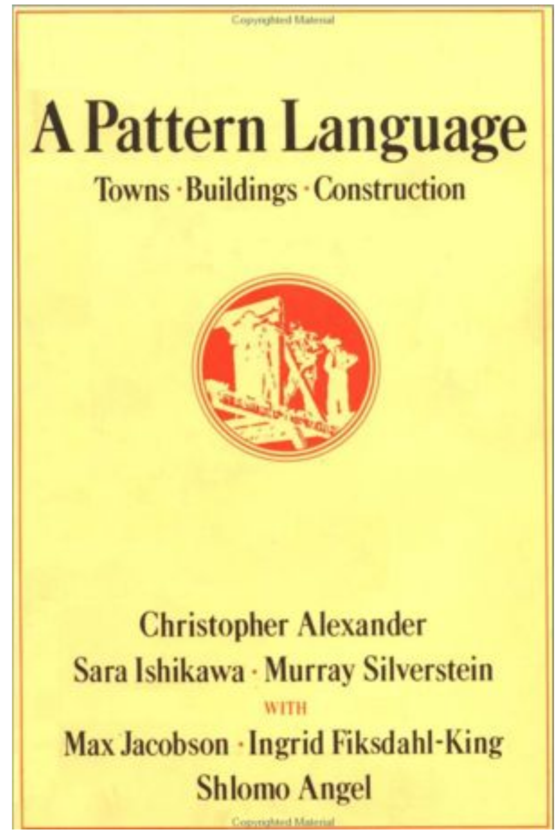
- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns

Elements of a pattern language



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!!!



Alexander's Pattern Language (95-103)

- Layout the overall arrangement of a group of buildings: the height and number of these buildings, the entrances to the site, main parking areas, and lines of movement through the complex.
- 95. Building Complex
- 96. Number of Stories
- 97. Shielded Parking
- 98. Circulation Realms
- 99. Main Building
- 100. Pedestrian Street
- 101. Building Thoroughfare
- 102. Family of Entrances
- 103. Small Parking Lots

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 entran 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).

Family of Entrances

http://www.intbau.org/Images/Steele/Badran5a.jpg

Elements of a Pattern - 1

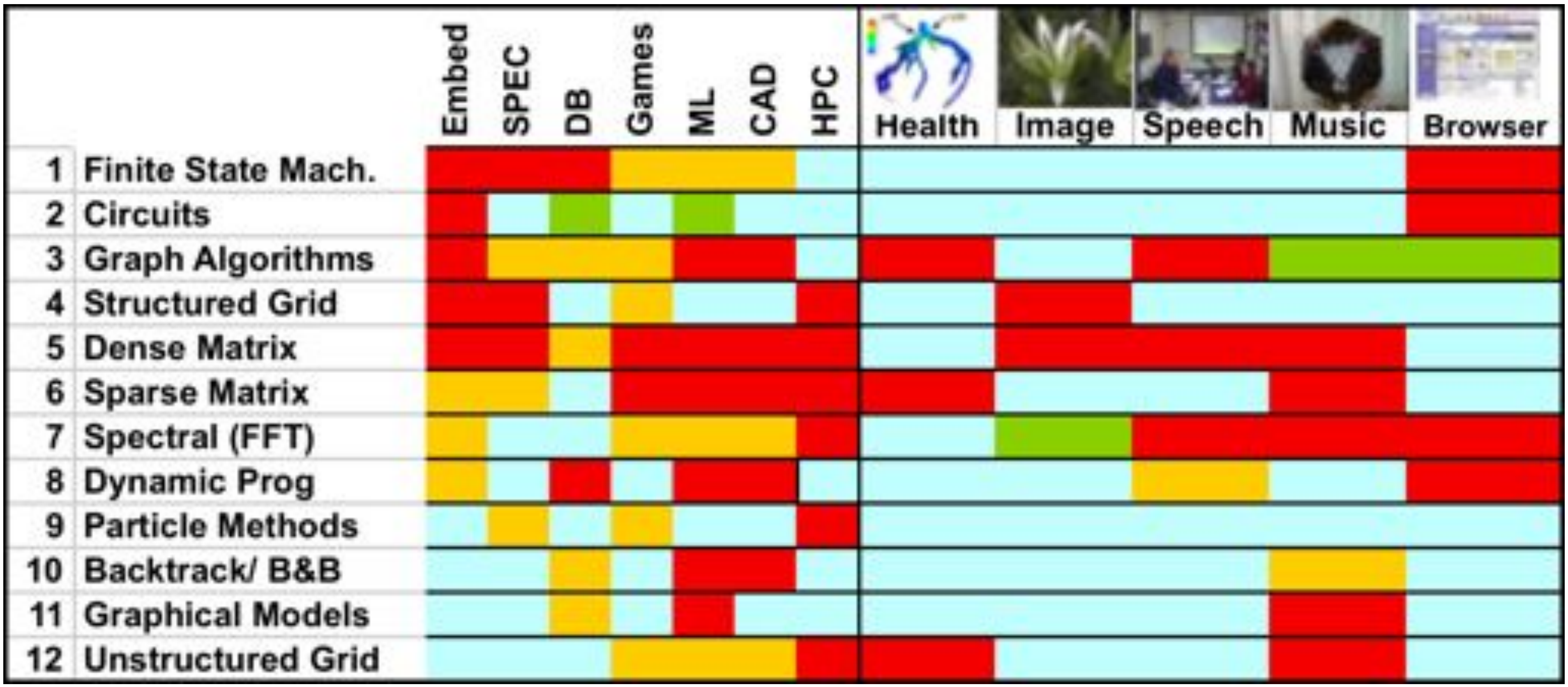
- Name
  - It must have a meaningful name useful to remember the pattern and when it is used.
- Problem
  - A statement of the problem … a one-line preamble and the problem stated as a question.
- Context
  - The conditions under which the problem occurs. Defines when the pattern is applicable and the configuration of the system before the pattern is applied.
- Forces
  - A description of the relevant *forces* and constraints and how they interact/conflict with one another and with goals we wish to achieve. Defines the tension that characterizes a problem.

Elements of a Pattern - 2

- Solution
  - Instructions used to solve the problem. When done right, it resolves the tension defined in the forces section; flowing from the context and forces. We also define the new context for the system following application of the pattern.
- Invariant
  - What must be invariantly true for this pattern to work. May be stated in the form of Precondition, Invariant, Post-condition
- Examples
  - Examples to help the reader understand the pattern.
- Known Uses and frameworks
  - Cases where the pattern was used; preferably with literature references.
- Related Patterns
  - How does this pattern fit-in or work-with the other patterns in the pattern language.

*PLPP is the pattern language from Mattson, Sanders and Massingill

Computational Patterns
(Red Hot → Blue Cool)



Patterns for Parallel Programming

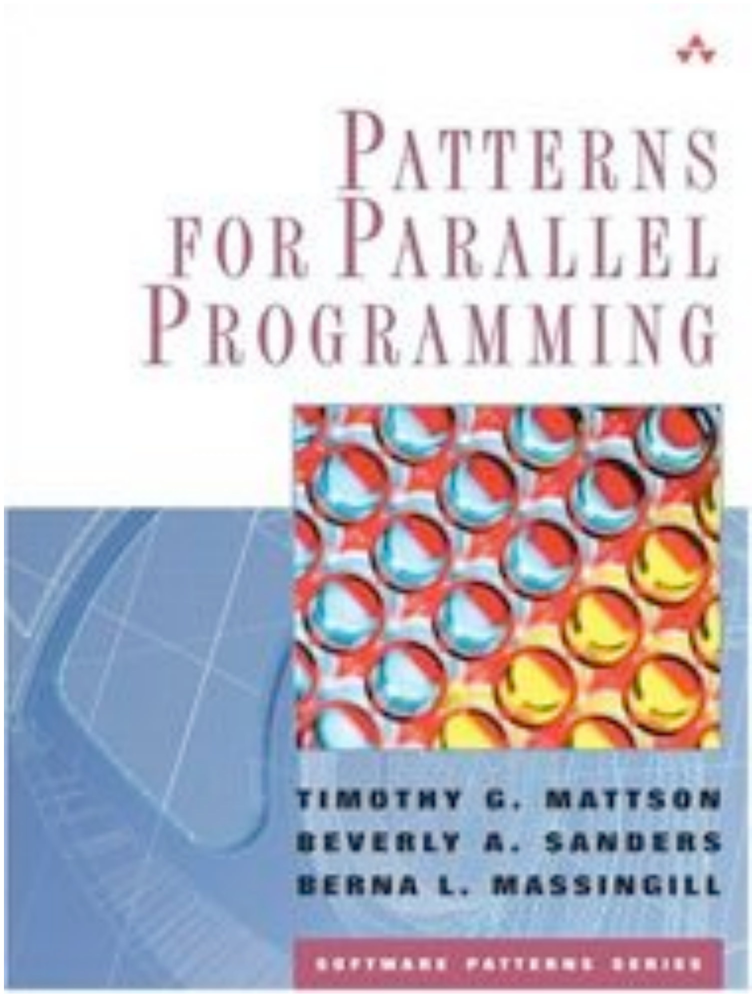
• PLPP is the first attempt to develop a complete *pattern language* for parallel software development.

• PLPP is a great model for a pattern language for parallel software

• PLPP mined scientific applications that utilize a monolithic application style

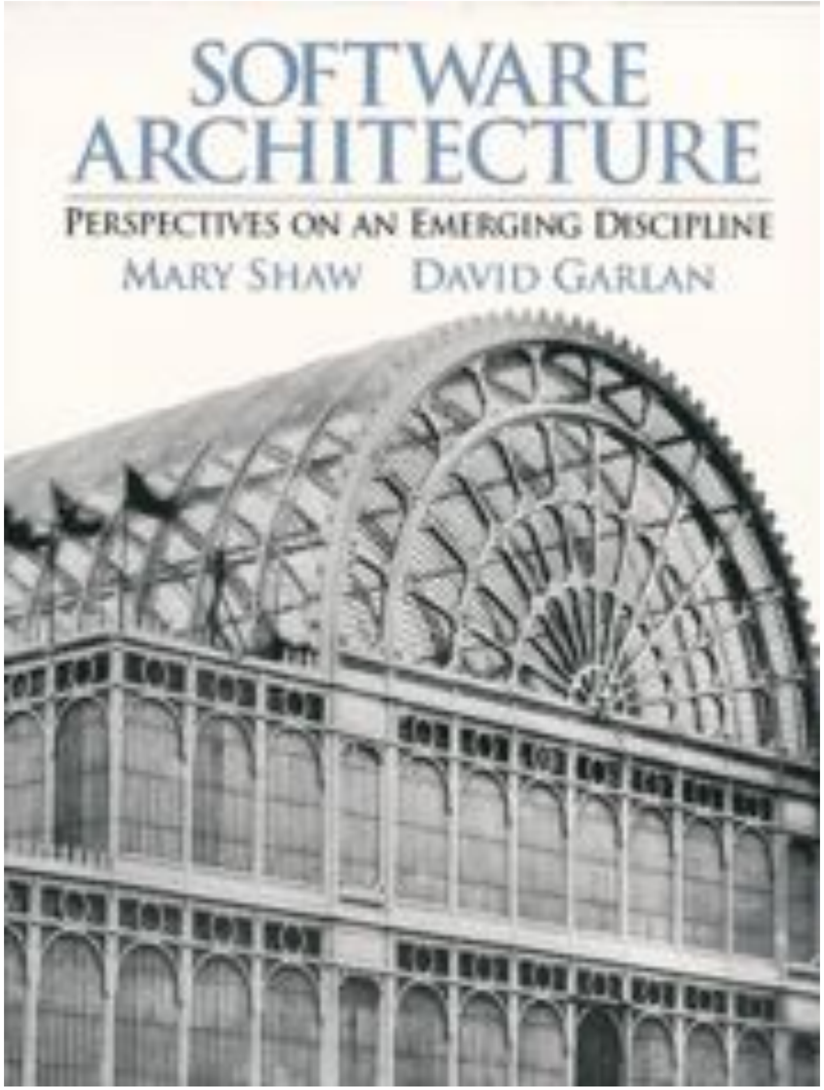
•PLPP doesn't help us much with horizontal composition

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

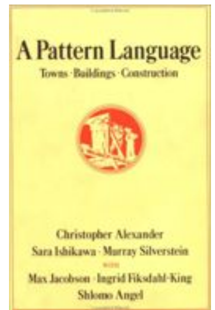


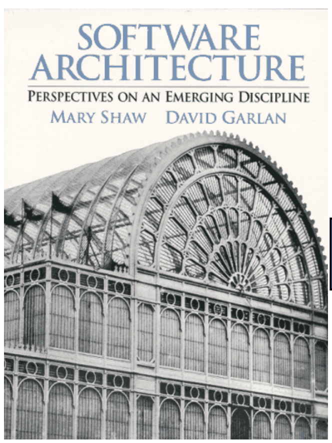
Structural programming patterns

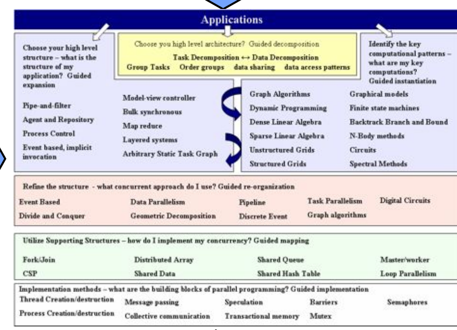
- In order to create more complex software it is necessary to compose programming patterns
- For this purpose, it has been useful to induct a set of patterns known as "architectural styles"
- Examples:
  - pipe and filter
  - event based/event driven
  - layered
  - Agent and repository/ blackboard
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

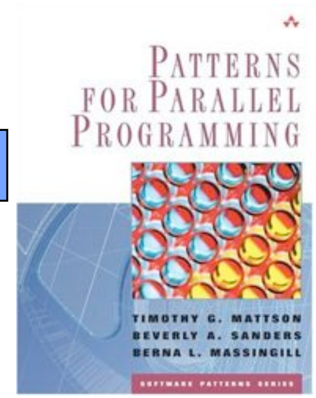


Put it all together









Our Pattern Language 2.0: Keutzer and Mattson

Applications

Productivity Layer

Choose your high level structure – what is the structure of my application? Guided expansion

- Pipe-and-filter
- Agent and Repository
- Process Control
- Event based, implicit invocation
- Model-view controller
- Iteration
- Map reduce
- Layered systems
- Arbitrary Static Task Graph

Choose you high level architecture? Guided decomposition

Task Decomposition ↔ Data Decomposition

Group Tasks   Order groups   data sharing   data access

Identify the key computational patterns – what are my key computations? Guided instantiation

- 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

Refine the structure - what concurrent approach do I use? Guided re-organization

- Event Based
- Divide and Conquer
- Data Parallelism
- Geometric Decomposition
- Pipeline
- Discrete Event
- Task Parallelism
- Graph algorithms
- Digital Circuits

Efficiency Layer

Utilize Supporting Structures – how do I implement my concurrency? Guided mapping

- Fork/Join
- CSP
- Distributed Array
- Shared Data
- Shared Queue
- Shared Hash Table
- SPMD
- Master/worker
- Loop Parallelism
- BSP

Implementation methods – what are the building blocks of parallel programming? Guided implementation

- Thread Creation/destruction
- Process Creation/destruction
- Message passing
- Collective communication
- Speculation
- Transactional memory
- Barriers
- Mutex
- Semaphores

Our Pattern Language 2.0

Applications

Choose your high level structure – what is the structure of my application? Guided

Choose you high level architecture? Guided decomposition

Task Decomposition ↔ Data Decomposition

Group Tasks Order groups data sharing data access

Identify the key computational patterns – what are my key computations?

Productivity Layer

Garlan and Shaw Architectural Styles

Agent and Repository

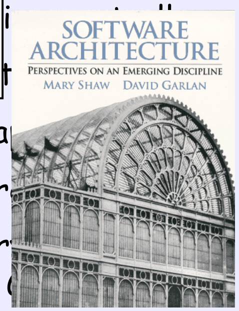
Process Control

Event based, implicit invocation

Ma

Layer

Arbitrar



Graph Algorithms

Dynamic Programming

Berkeley View 13 dwarfs

ktrack Branch and Bound

N-Body methods

Circuits

Spectral Methods

Refine the structure - what concurrent approach do I use? Guided re-organization

Event Based

Divide and Conquer

Data P

Geometric

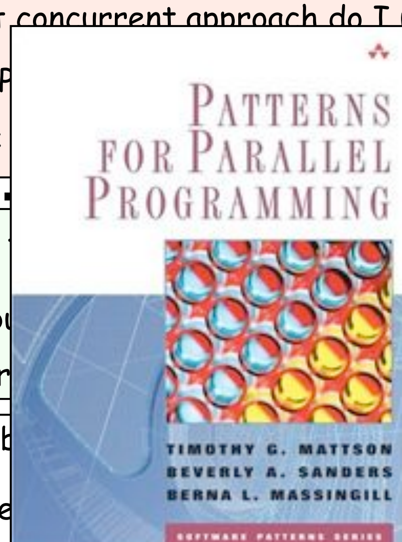
he

Event

Task Parallelism

Graph algorithms

Digital Circuits



Efficiency Layer

Utilize Supporting Structures

ncurrency? Guided mapping

Fork/Join

CSP

Distrib

Shar

Shared Queue

red Hash Table

SPMD

Master/worker

Loop Parallelism

BSP

Implementation methods – what are the b

rogramming? Guided implementation

Thread Creation/destruction

Process Creation/destruction

Message

Collective communication

culation

Transactional memory

Barriers

Mutex

Semaphores

Architecting Parallel Software

Decompose Tasks/Data

Order tasks   Identify Data Sharing and Access

Identify the Software Structure

- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Bulk Synchronous
- MapReduce
- Layered Systems
- Arbitrary Task Graphs

Identify the Key Computations

- Graph Algorithms
- Dynamic programming
- Dense/Spare Linear Algebra
- (Un)Structured Grids
- Graphical Models
- Finite State Machines
- Backtrack Branch-and-Bound
- N-Body Methods
- Circuits
- Spectral Methods

Pop Quiz: Software is More Like …

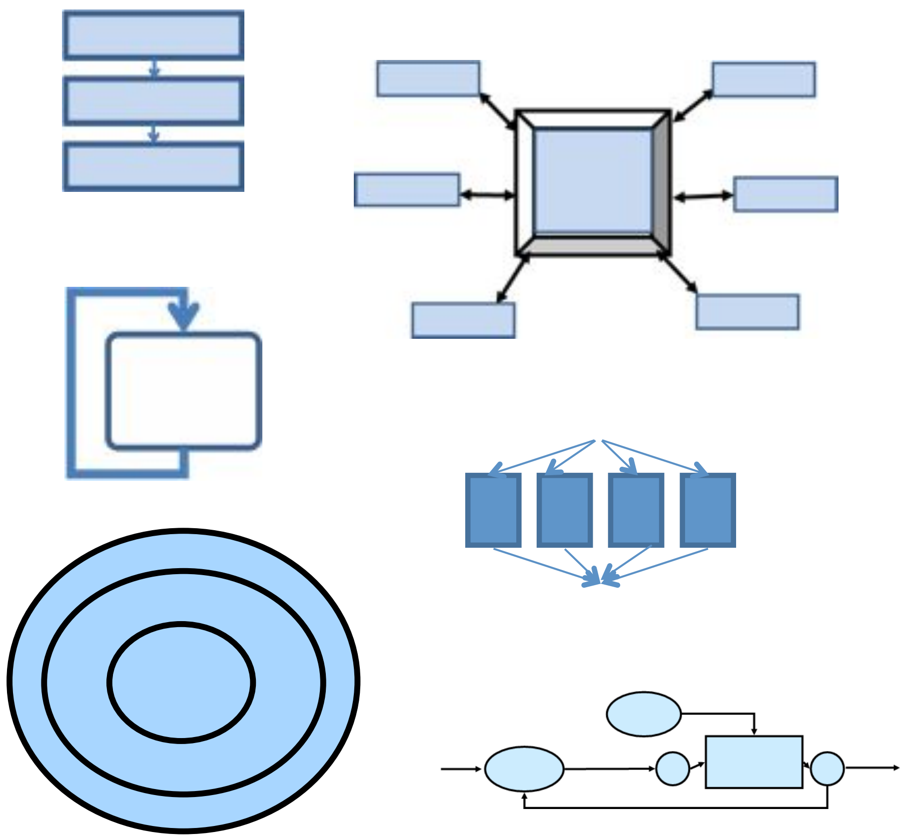
a) A building

b) A factory

Identify the SW Structure

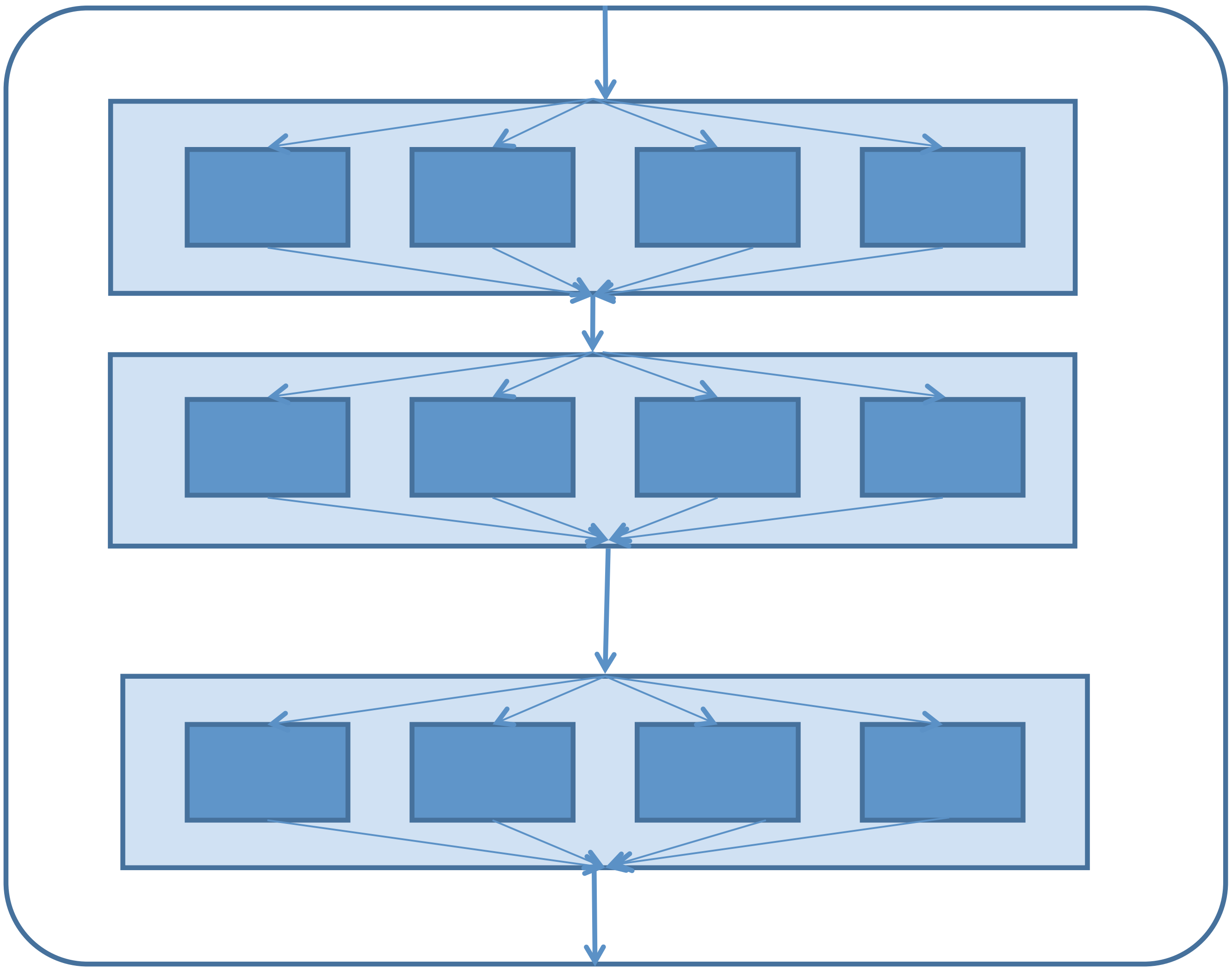
Structural Patterns

- Pipe-and-Filter
- Agent-and-Repository
- Event-based coordination
- Iterator
- MapReduce
- Process Control
- Layered Systems

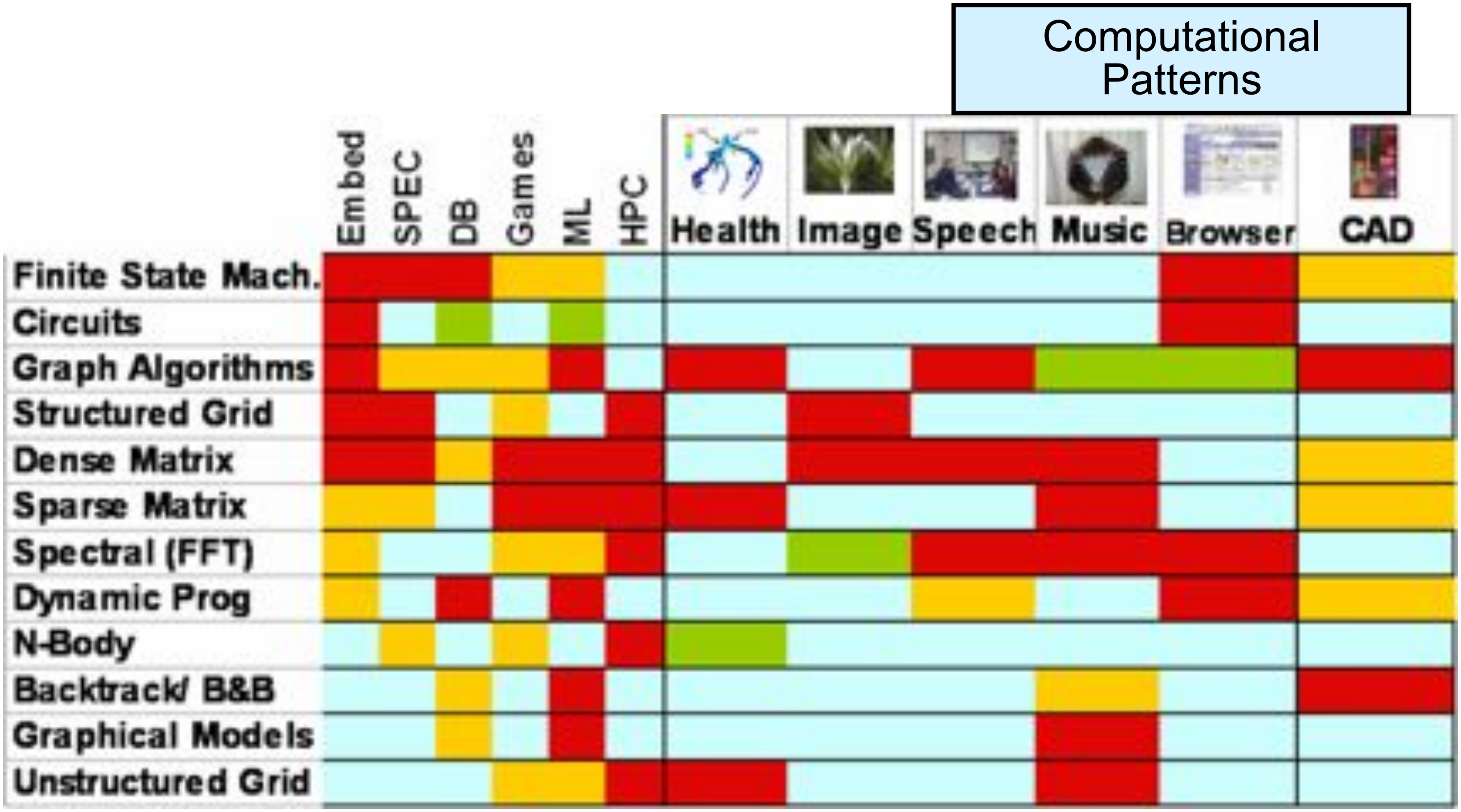


These define the structure of our software but they *do not describe* what is computed

Analogy: Layout of Factory Plant

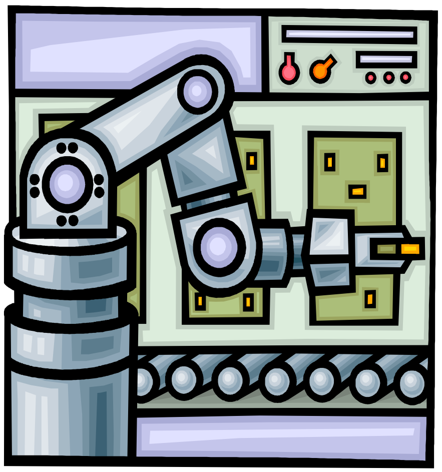
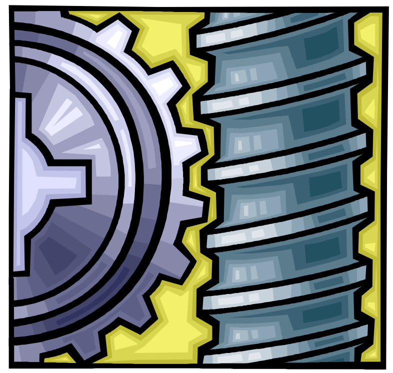
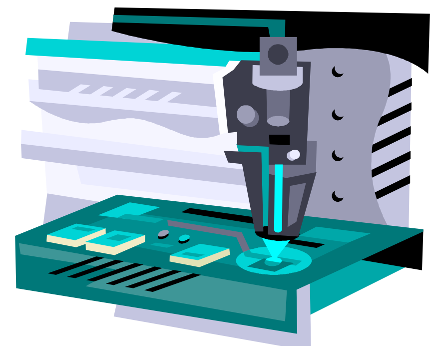
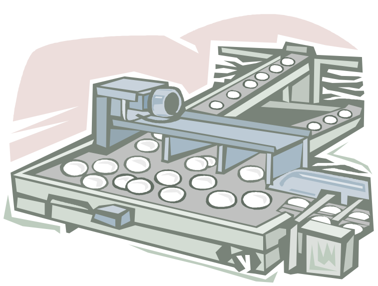
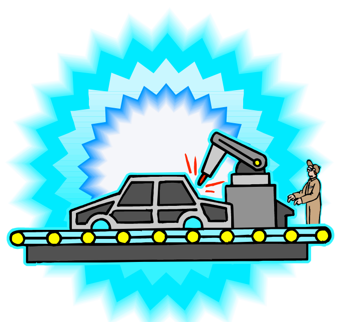
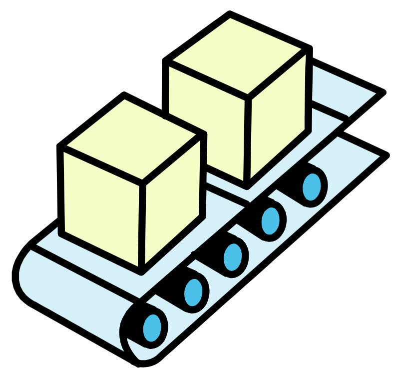
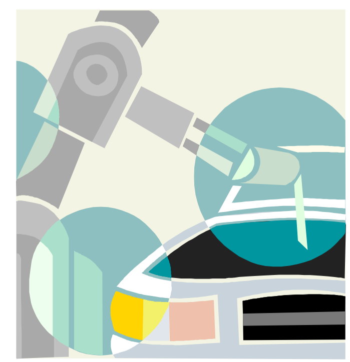


Identify Key Computations

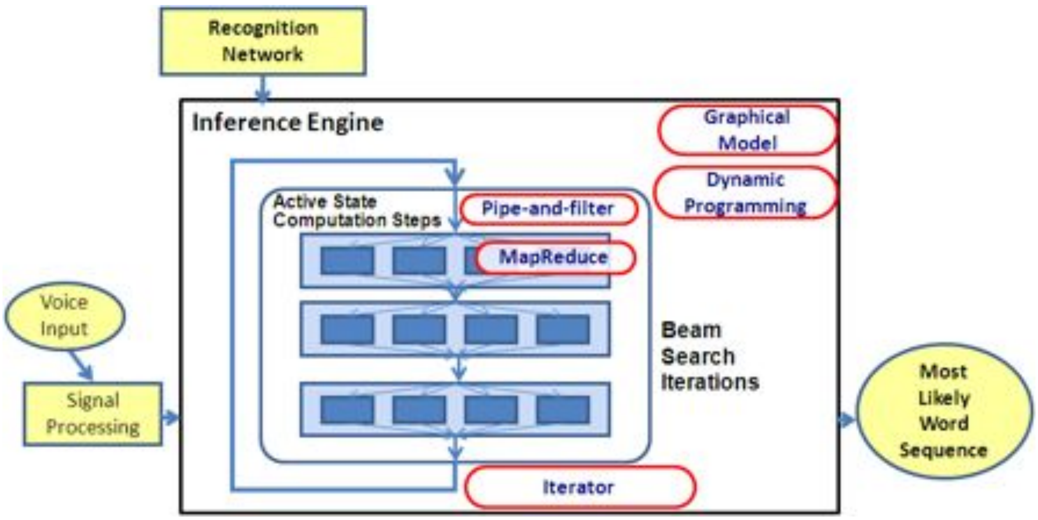


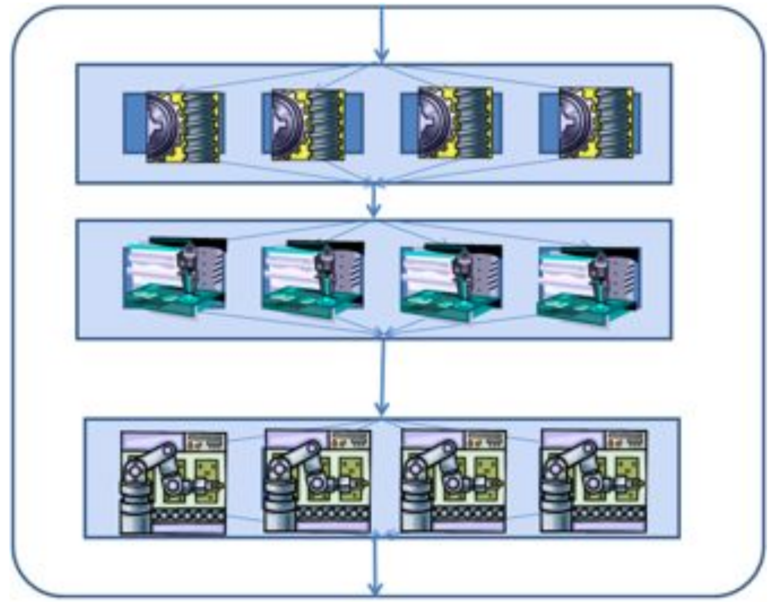
- Computational patterns describe the key computations but not how they are implemented

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

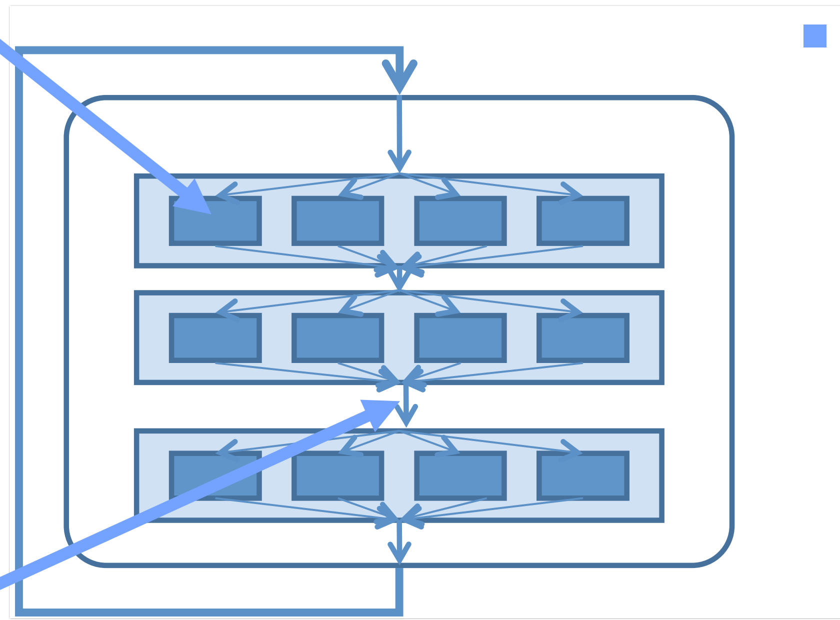
- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns

Inventory of Structural Patterns

- pipe and filter
- iterator
- MapReduce
- blackboard/agent and repository
- process control
- layered
- event-based coordination
- puppeteer
- (call-and-return/arbitrary task graph)

Elements of a structural pattern

- Components are where the computation happens



- A configuration is a graph of components (vertices) and connectors (edges)
- A structural patterns may be described as a familiy of graphs.

Connectors are where the communication happens

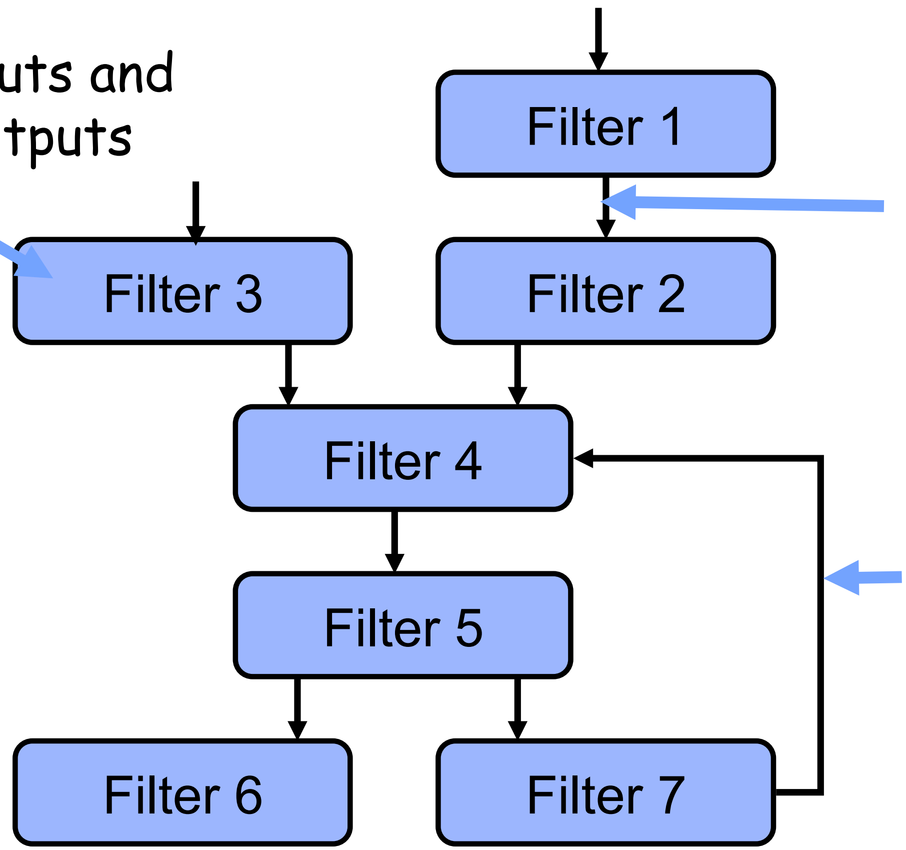
Pattern 1: Pipe and Filter

•Filters embody computation

•Only see inputs and produce outputs

•Pipes embody communication

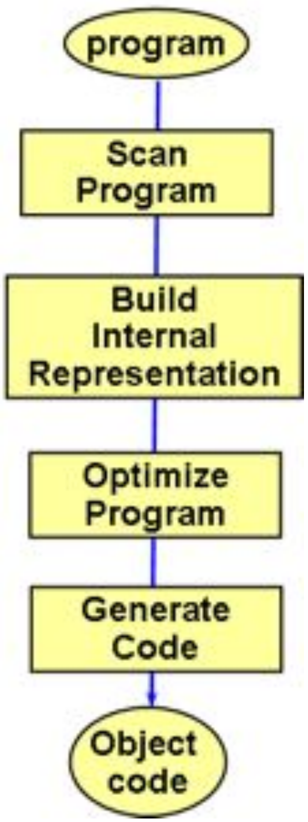
May have feedback



Examples?

Examples of pipe and filter

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



Compiler

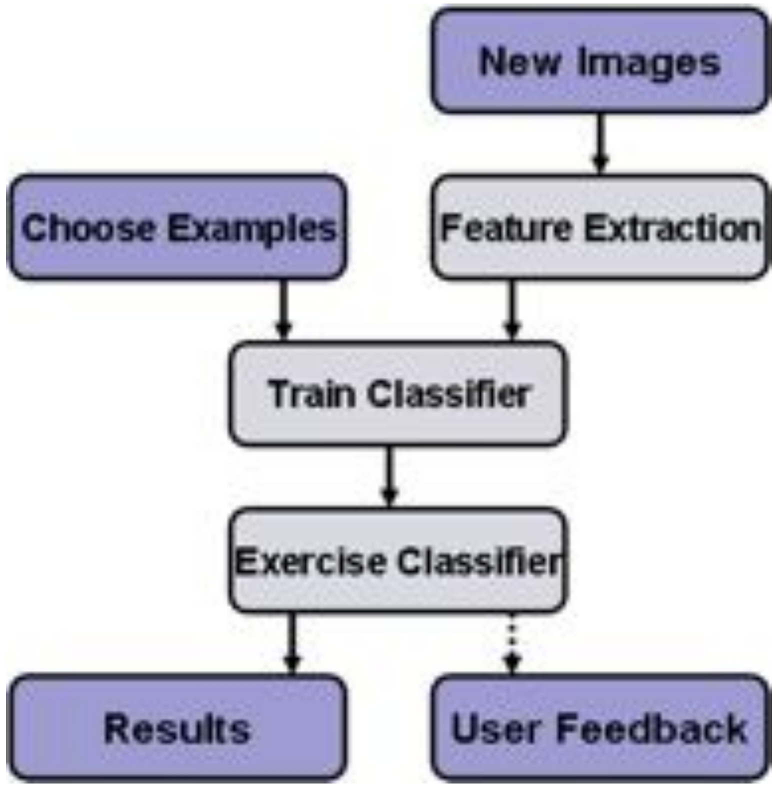
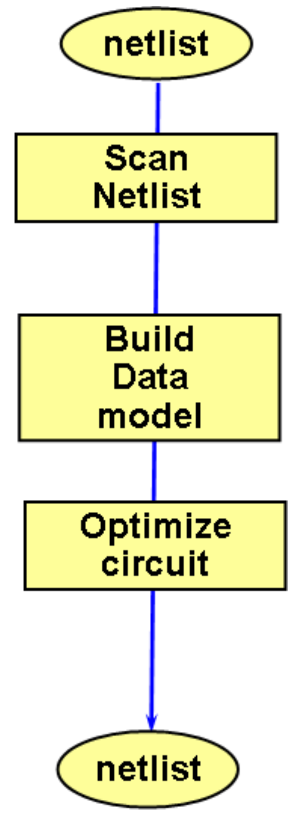
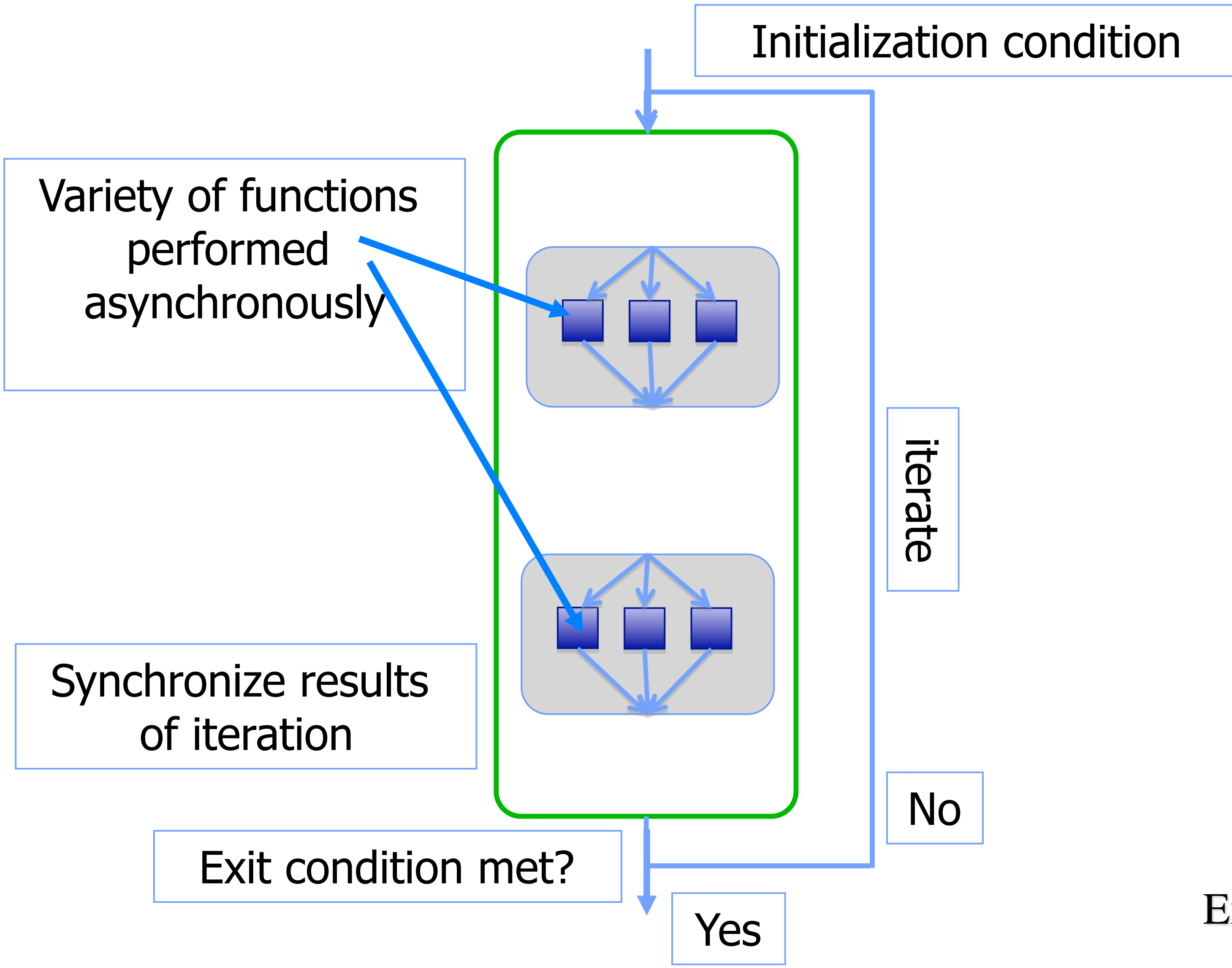


Image Retrieval System



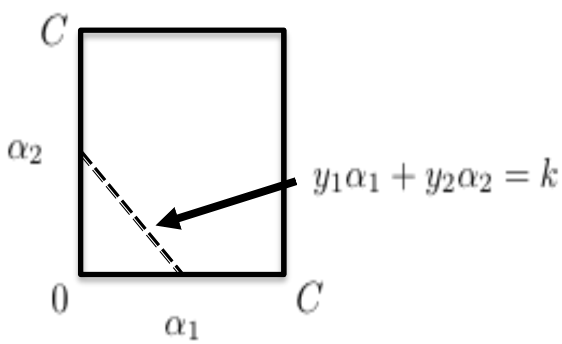
Logic optimizer

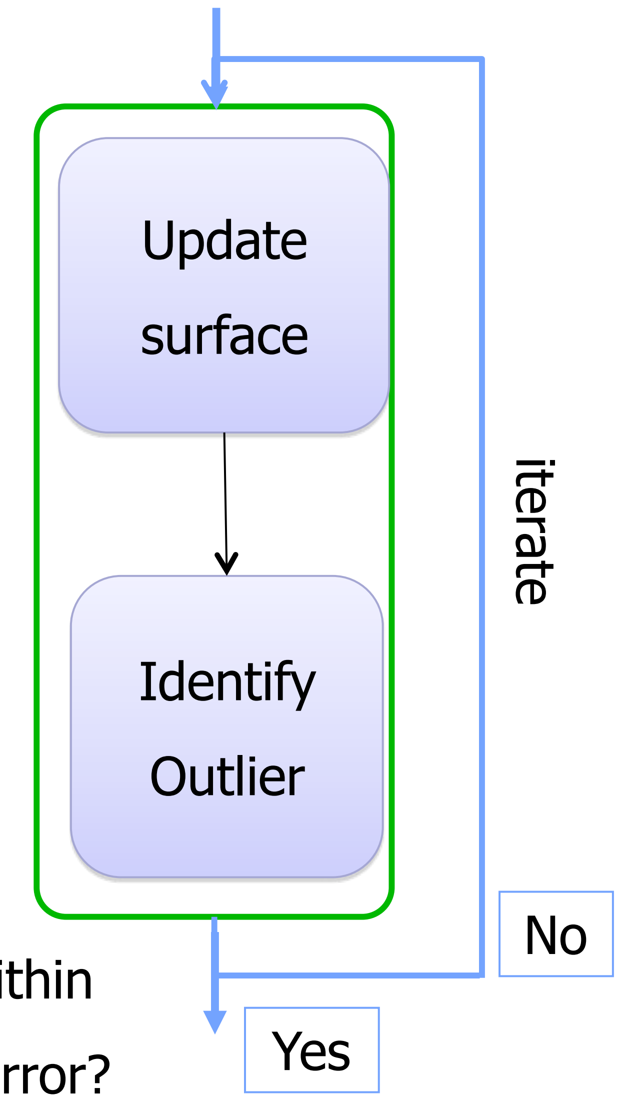
Pattern 2: Iterator Pattern



Examples?

Example of Iterator Pattern: Training a Classifier: SVM Training



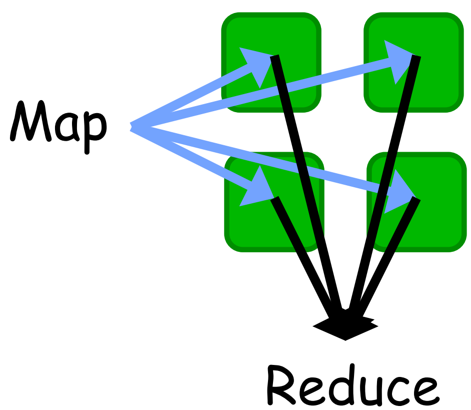


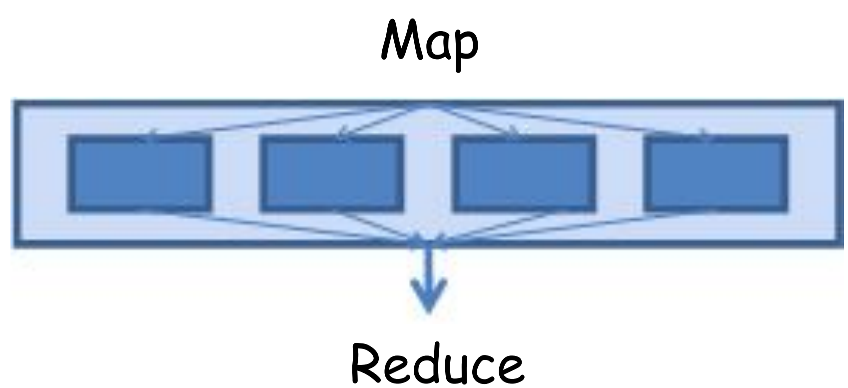
Iterator Structural Pattern

38

Pattern 3: MapReduce

- 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)





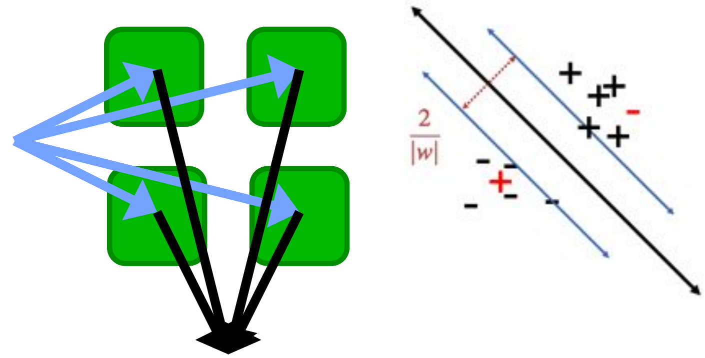
Examples?

Examples of Map Reduce

- General structure:

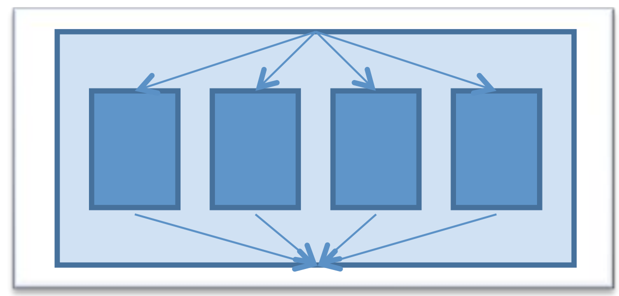
■ Map a computation across distributed data sets

■ Reduce the results to find the best/(worst), maxima/(minima)



Support-vector machines (ML)

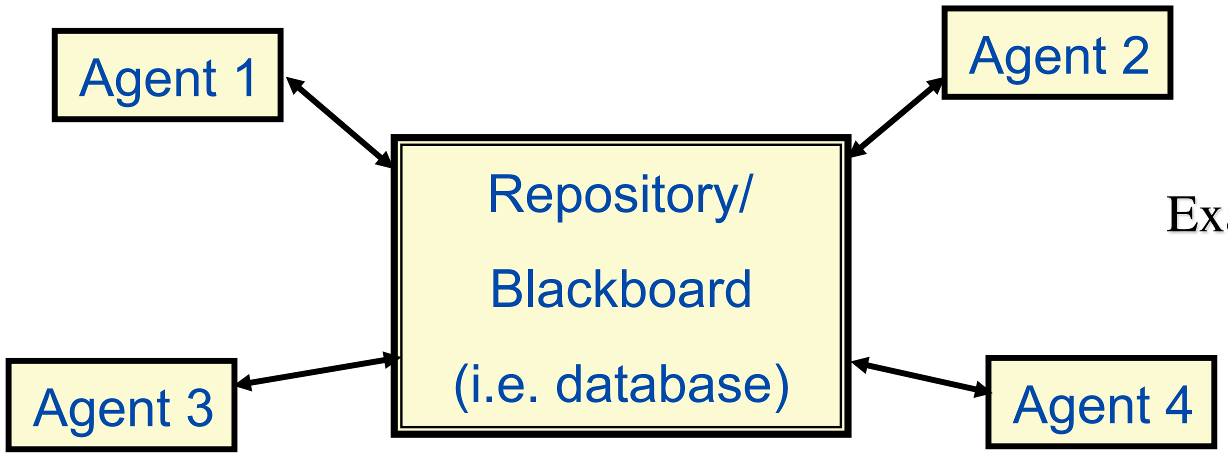
- Map to evaluate distance from the frontier
- Reduce to find the greatest outlier from the frontier



Speech recognition

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

Pattern 4: Agent and Repository



Examples?

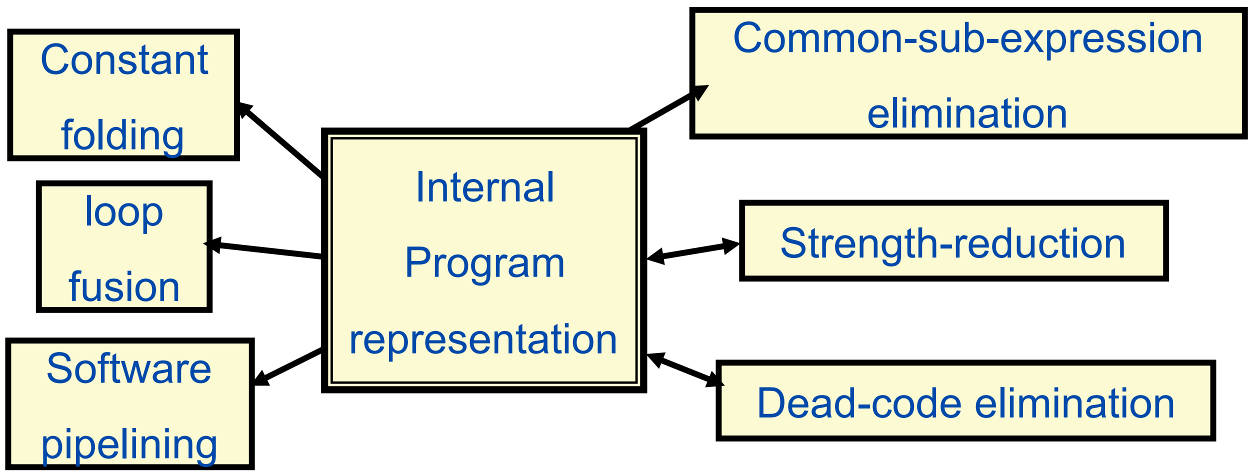
Agent and repository : Blackboard structural pattern

Agents cooperate on a shared medium to produce a result

Key elements:

- Blackboard: repository of the resulting creation that is shared by all agents (circuit database)
- Agents: intelligent agents that will act on blackboard (optimizations)
- Manager: orchestrates agents access to the blackboard and creation of the aggregate results (scheduler)

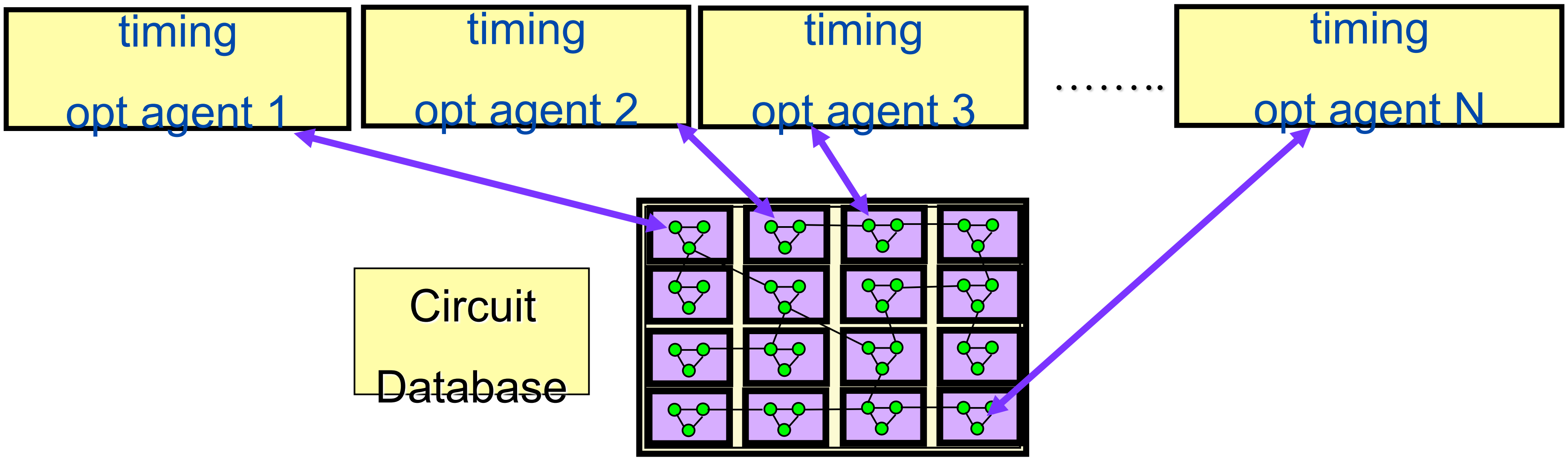
Example: Compiler Optimization



Optimization of a software program

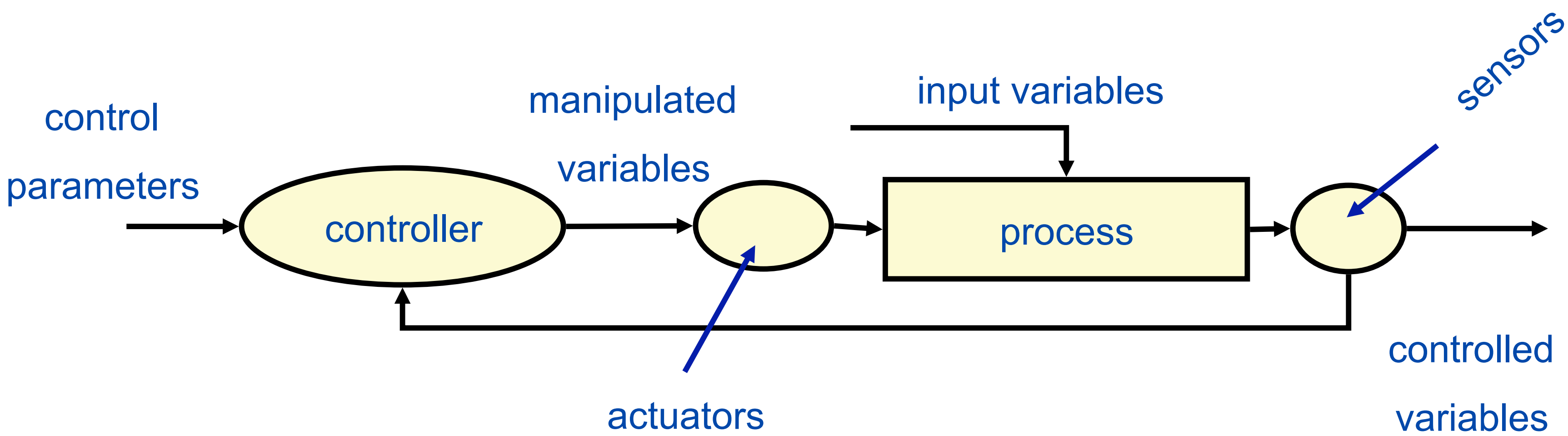
- Intermediate representation of program is stored in the repository
  - Individual agents have heuristics to optimize the program
- Manager orchestrates the access of the optimization agents to the program in the repository
  - Resulting program is left in the repository

Example: Logic Optimization



- Optimization of integrated circuits
- Integrated circuit is stored in the repository
- Individual agents have heuristics to optimize the circuitry of an integrated circuit
- Manager orchestrates the access of the optimization agents to the circuit repository
- Resulting optimized circuit is left in the repository

Pattern 5: Process Control

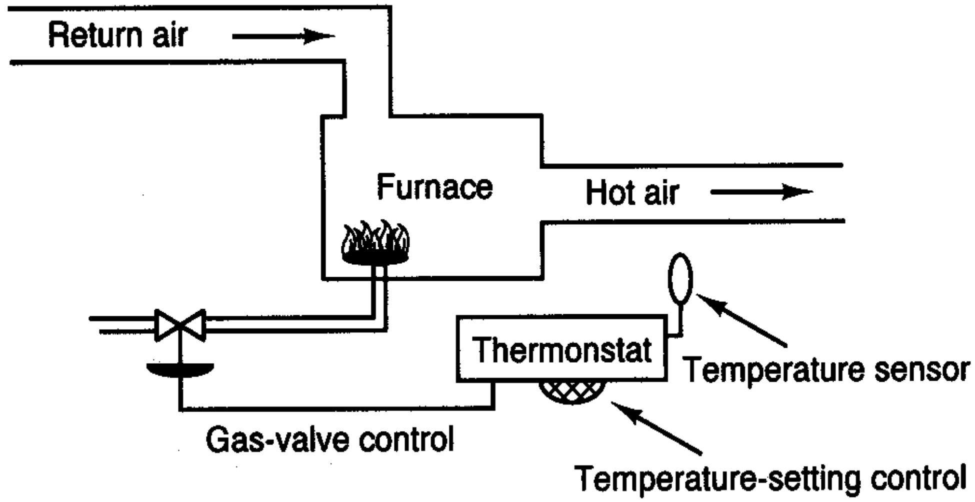


Source: Adapted from Shaw & Garlan 1996, p27-31.

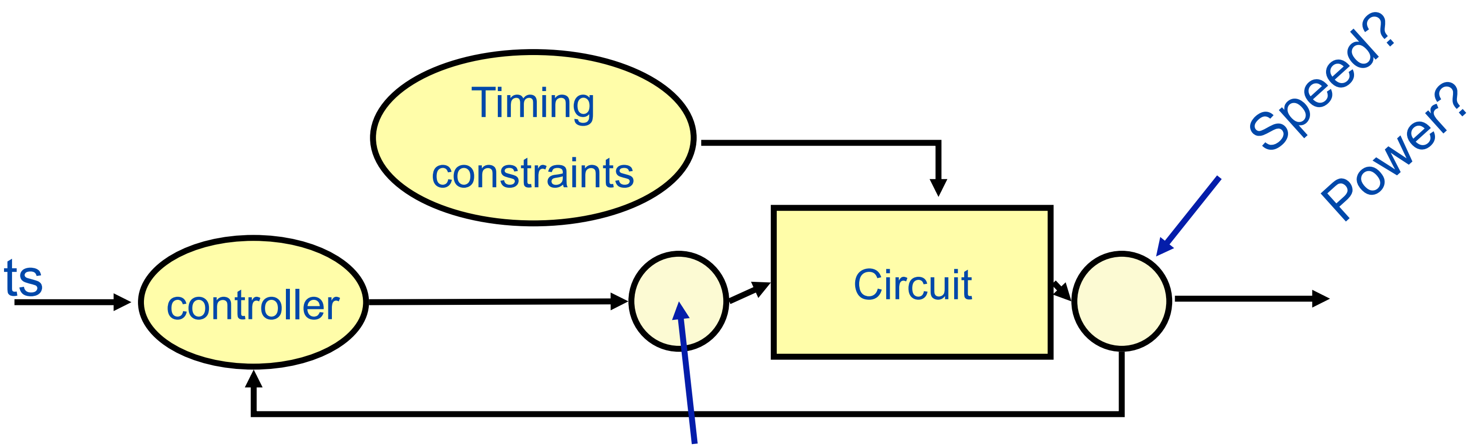
- Process control:
  - Process: underlying phenomena to be controlled/computed
  - Actuator: task(s) affecting the process
  - Sensor: task(s) which analyze the state of the process
  - Controller: task which determines what actuators should be effected

Examples?

Examples of Process Control



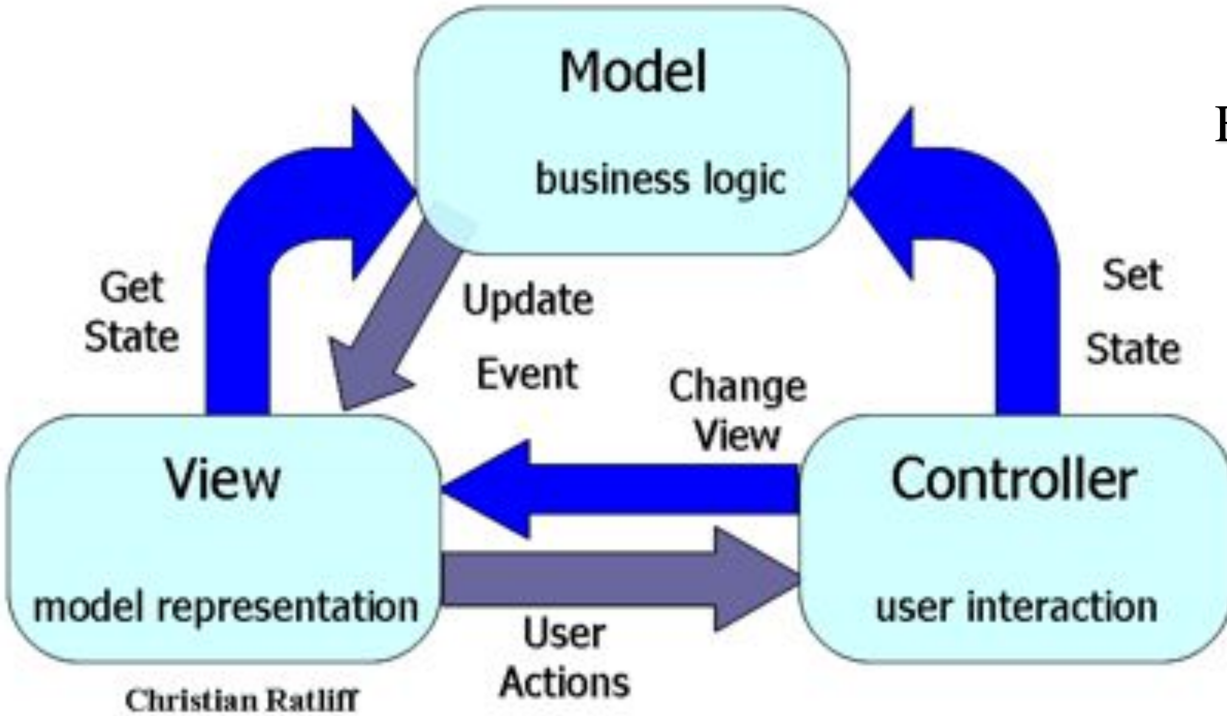
user
timing
constraints



Process control
structural pattern

Launching
transformations

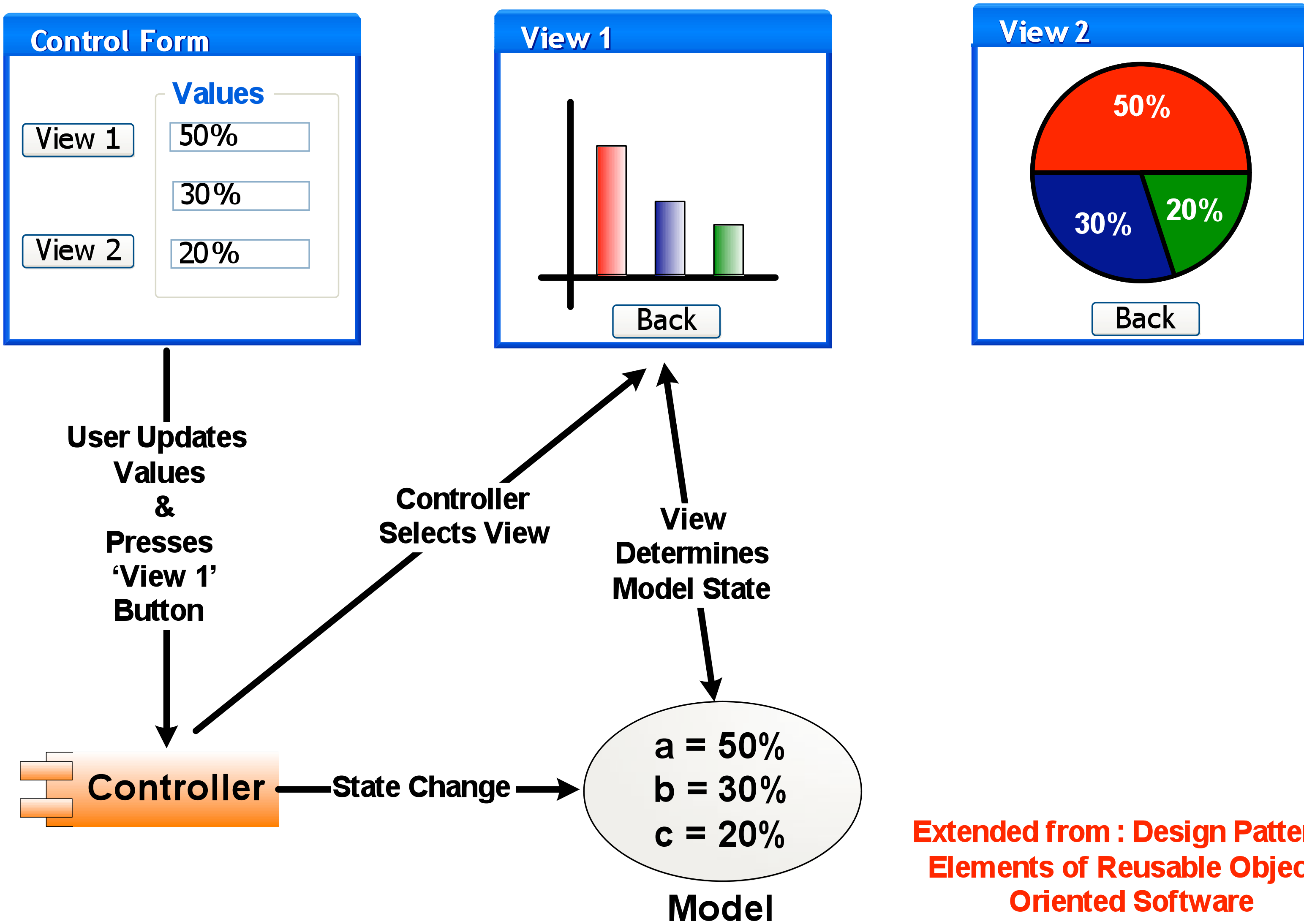
Pattern 6: Model-View-Controller



Examples?

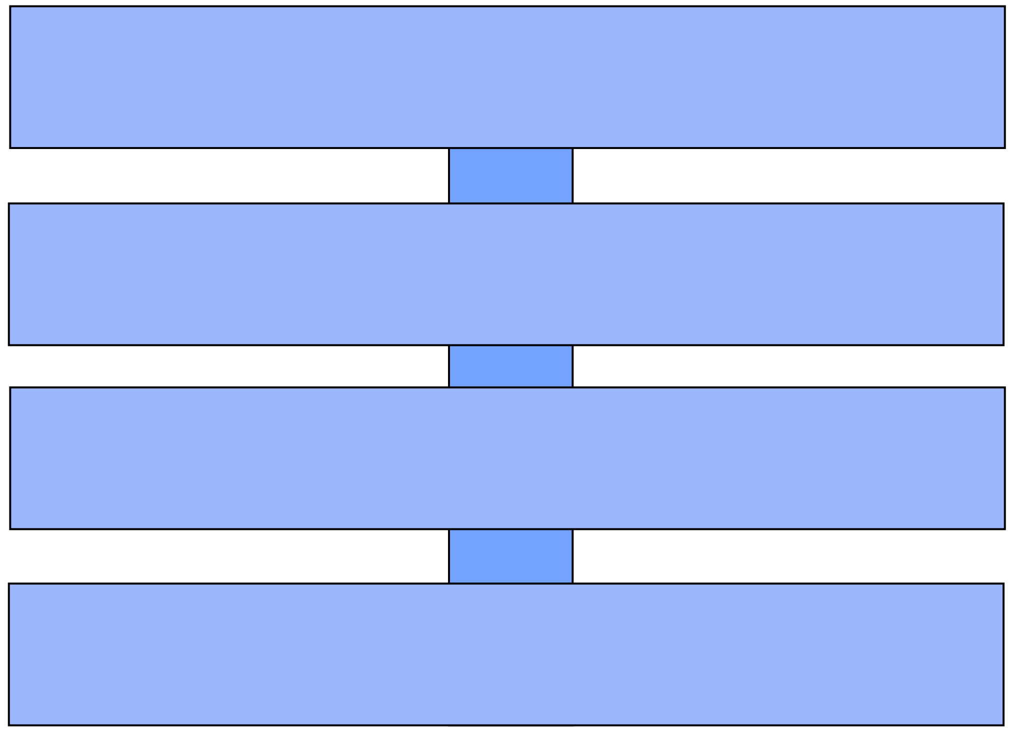
- Model: embodies the data and "intelligence" (aka business logic) of the system
- Controller: captures all user input and translates it into actions on the model
- View: renders the current state of the model for user

Example of Model-View Controller



Extended from : Design Patterns Elements of Reusable Object - Oriented Software

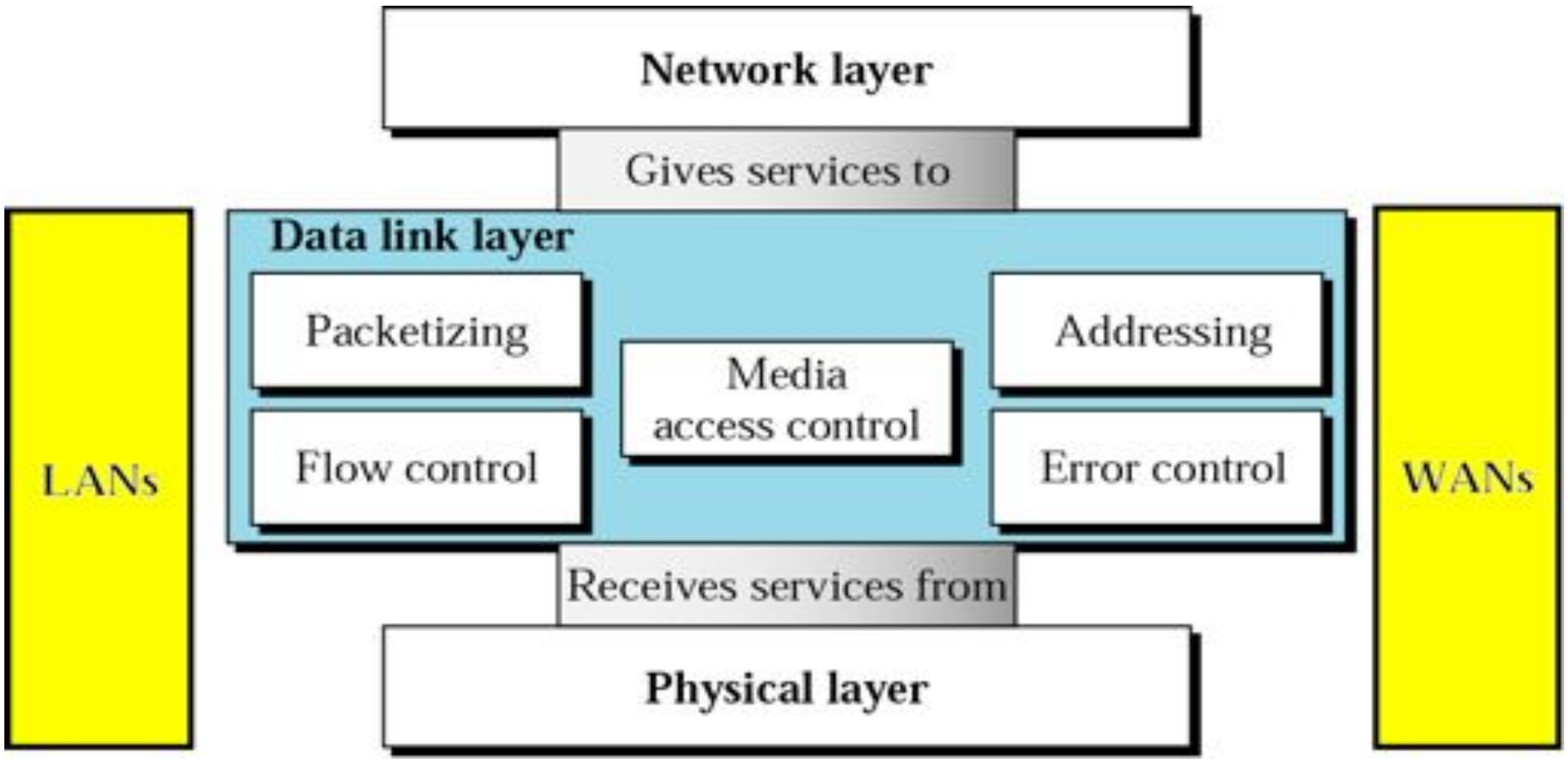
Pattern 7: Layered Systems



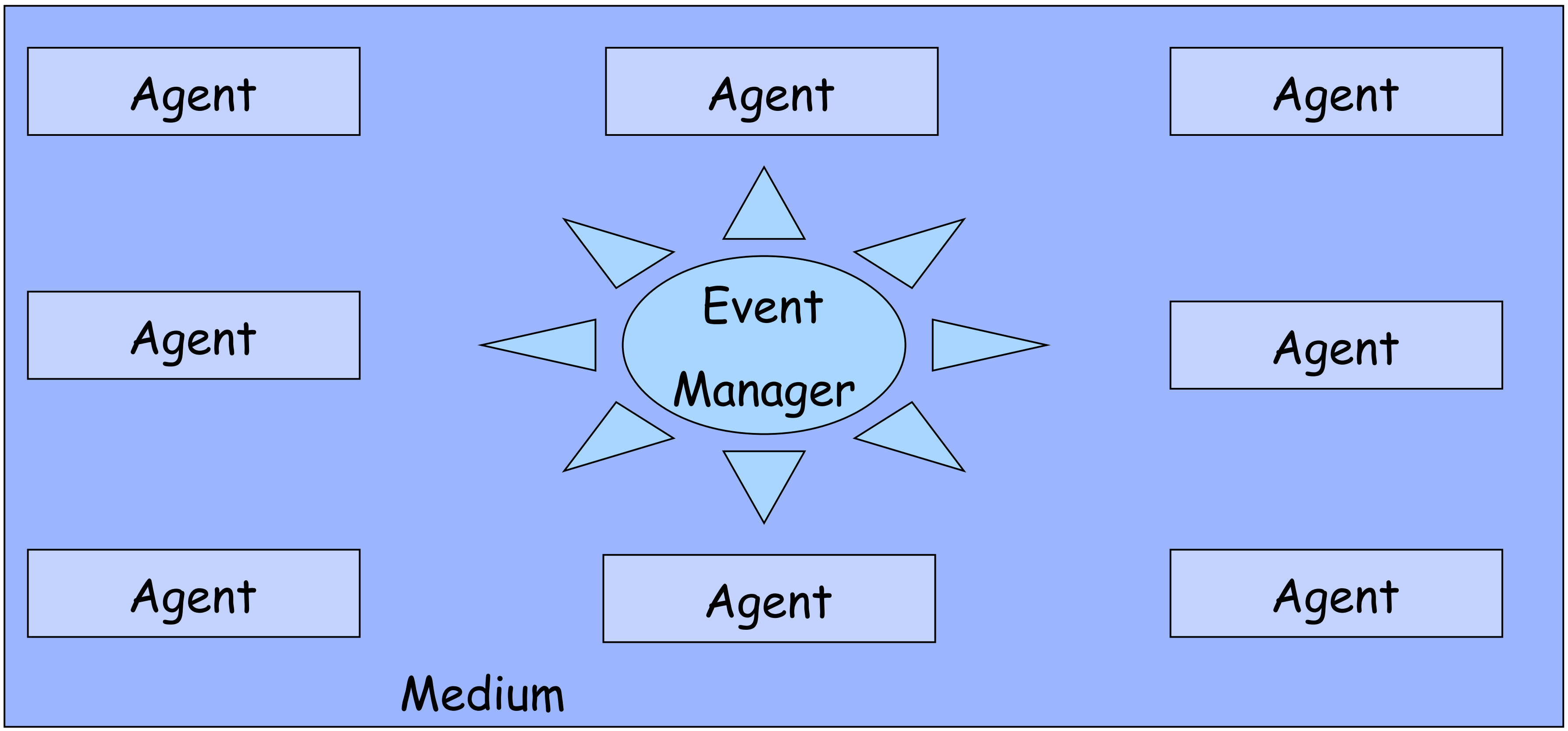
- Individual layers are big but the interface between two adjacent layers is narrow
- Non-adjacent layers cannot communicate directly.

Examples?

Example: ISO Network Protocol



Pattern 8: Event-based Systems

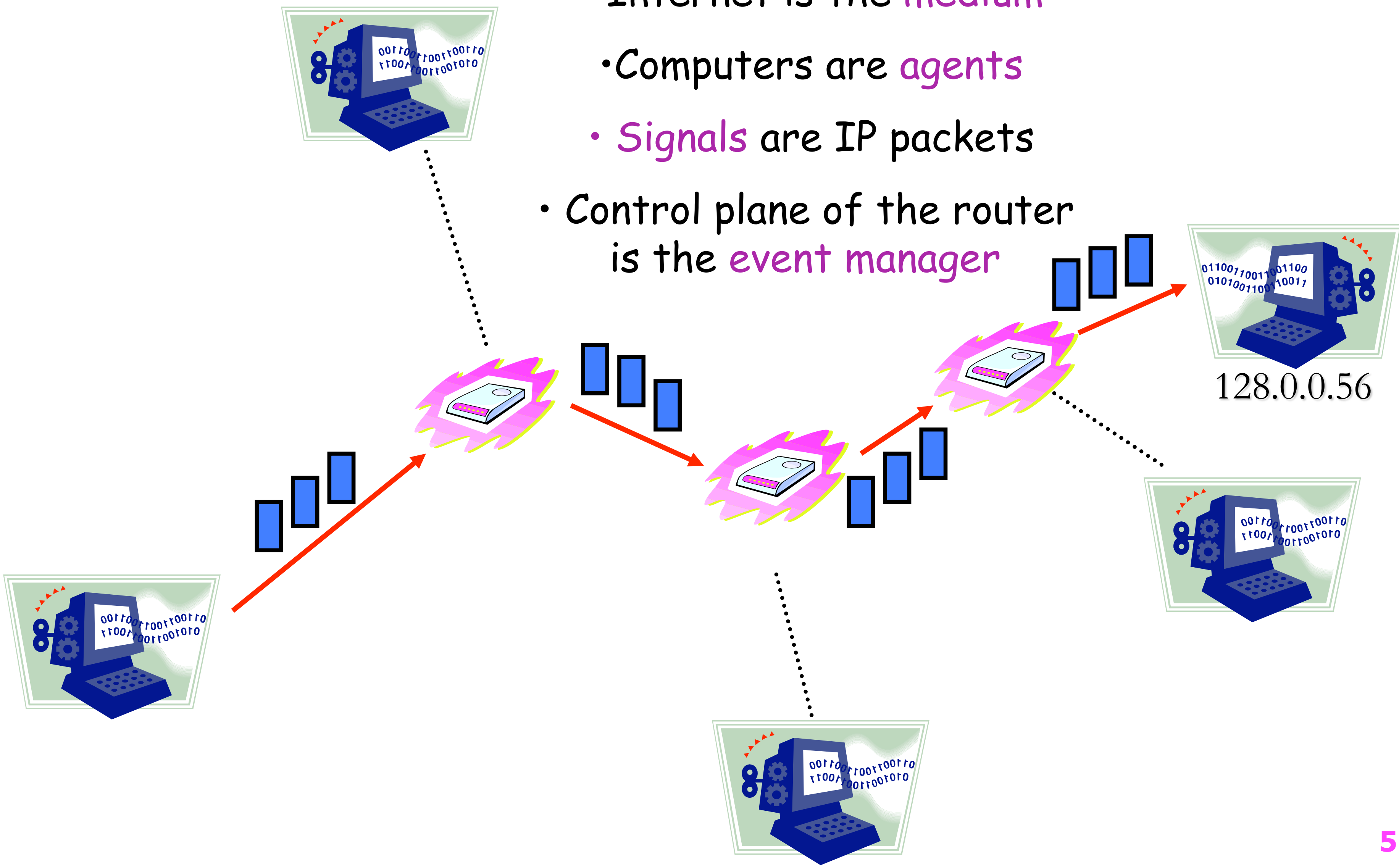


- Agents interact via events/signals in a medium
- Event manager manages events
- Interaction among agents is dynamic – no fixed connection

Examples?

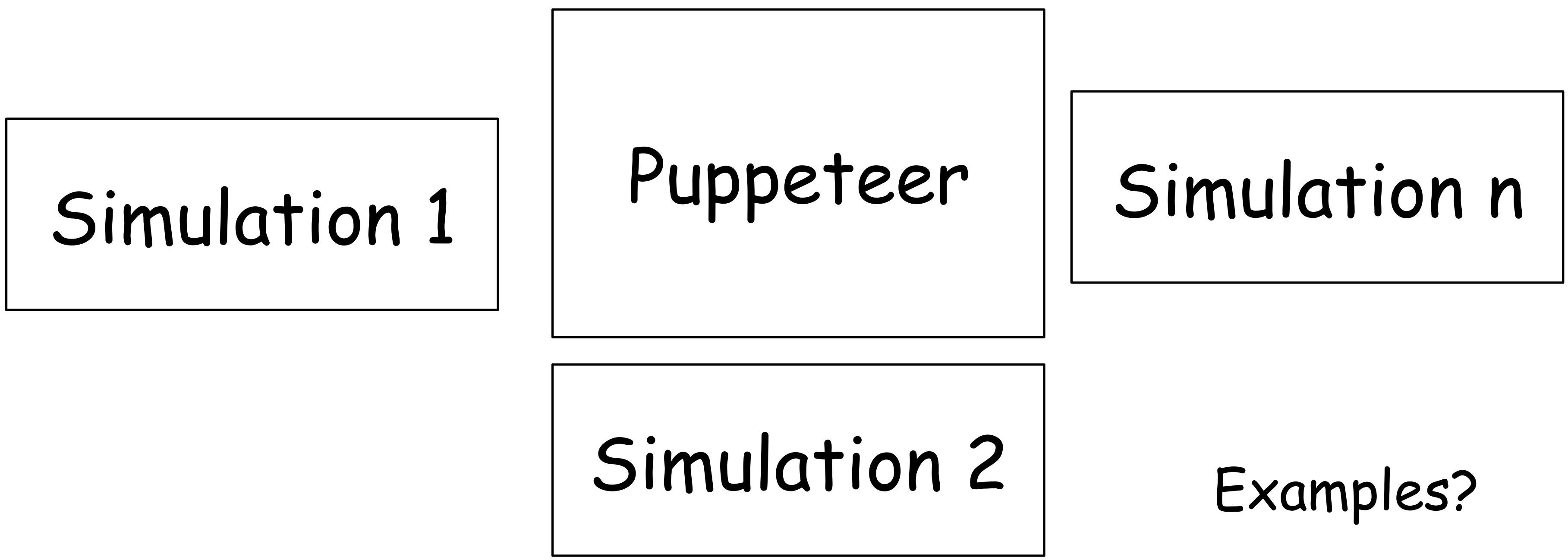
Example: The Internet

- Internet is the medium
- Computers are agents
- Signals are IP packets
- Control plane of the router is the event manager



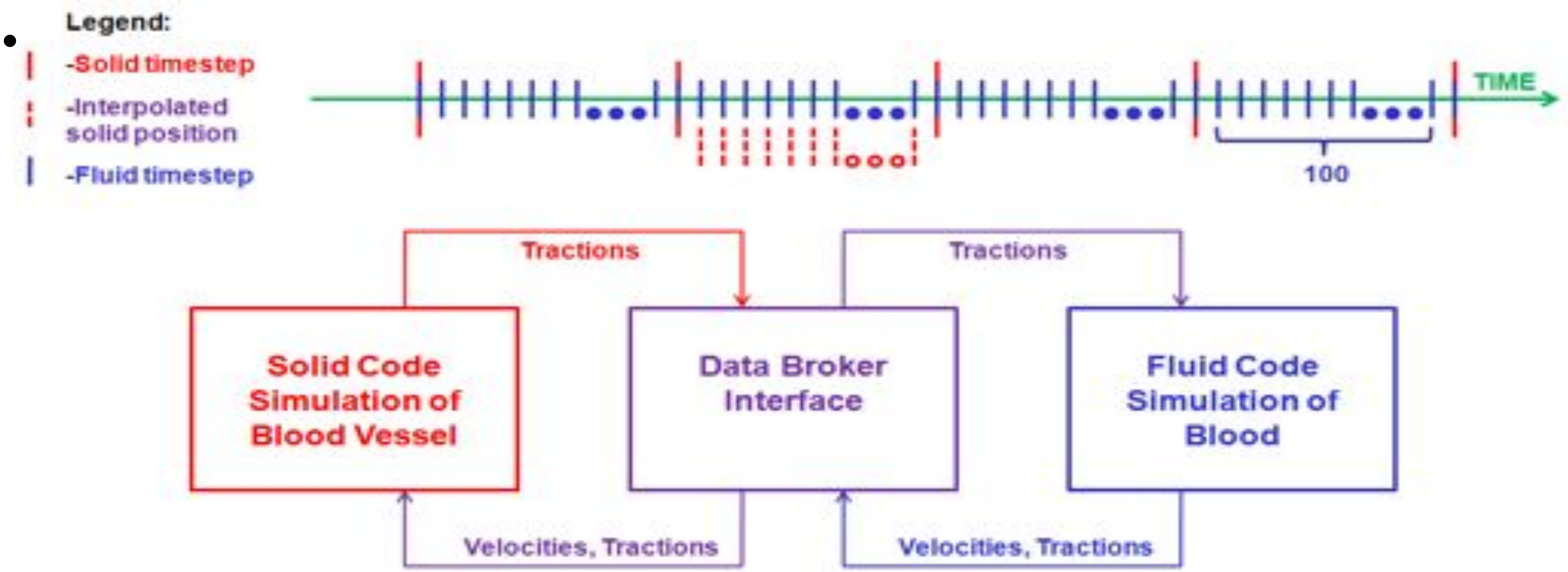
Pattern 9: Puppeteer

- Need an efficient way to manage and control the interaction of multiple simulators/computational agents
- **Puppeteer Pattern** – guides the interaction between the simulation codes to guarantee correctness of the overall simulation
- Difference with agent and repository?
    - No central repository
    - Data transfer between simulators

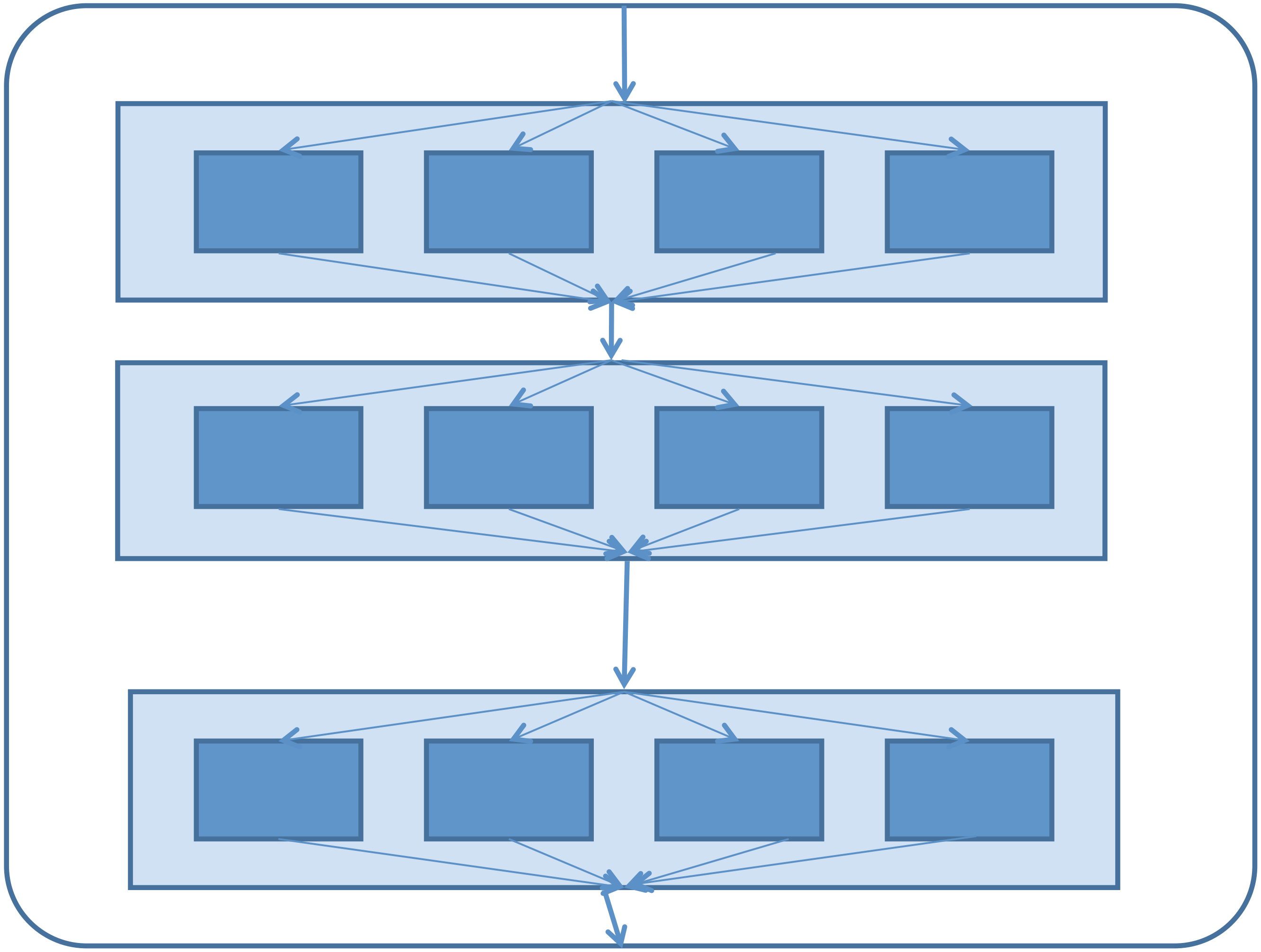


Examples?

Overall Computation

- Modeling of blood moving in blood vessels
- The computation is structured as a controlled interaction between solid (blood vessel) and fluid (blood) simulation codes
- The two simulations use different data structures and the number of iterations for each simulation code varies
- Need an efficient way to manage and control the interaction of the two codes
- 

Remember the Analogy: Layout of Factory Plant



- We have only talked about structure. We haven't described computation.

Architecting Parallel Software

Decompose Tasks

•Group tasks

•Order Tasks

Decompose Data

•Identify data sharing

•Identify data access

Identify the Software Structure

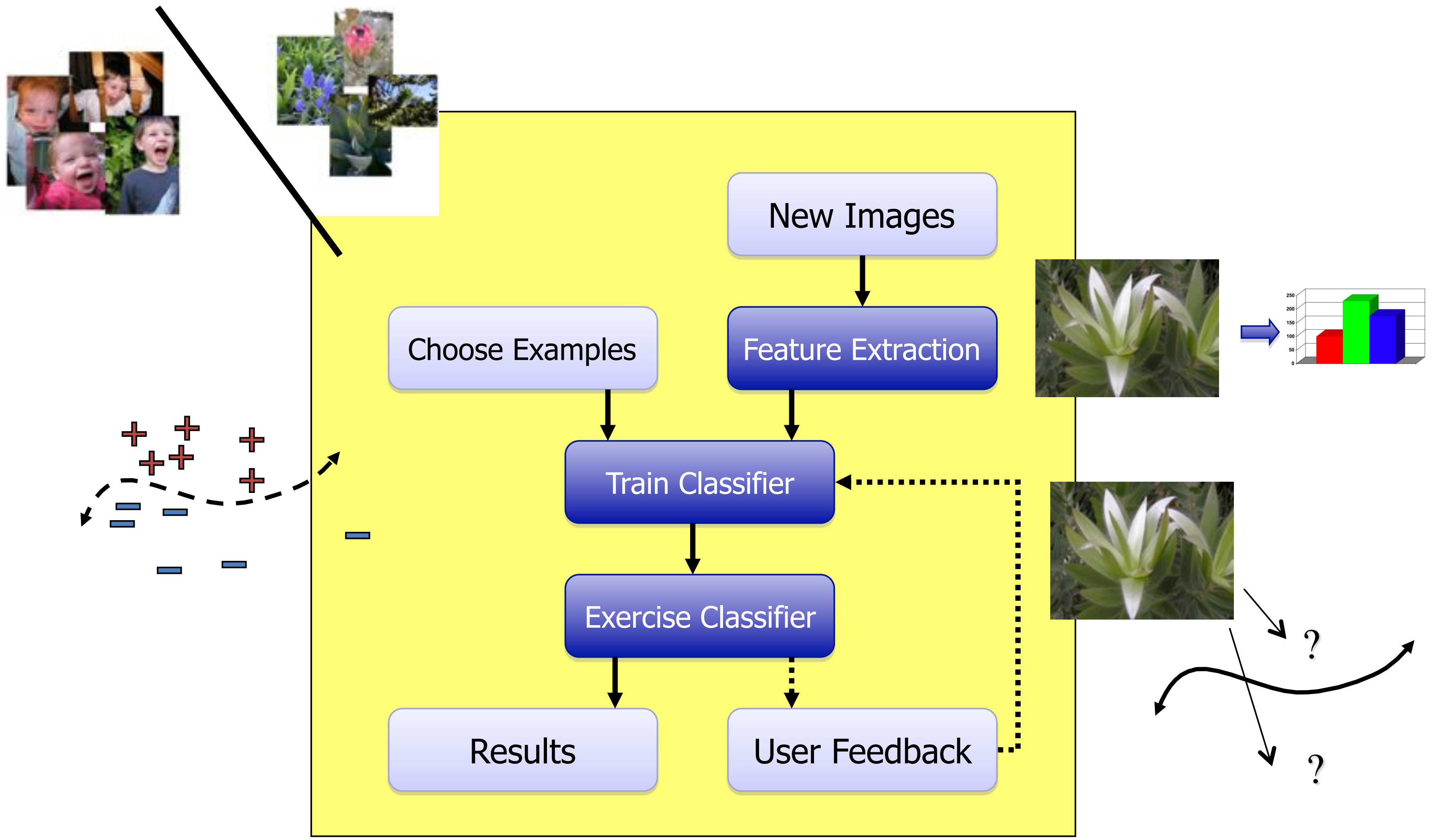
Identify the Key Computations

Friday: Computational Patterns of Parallel Programming (James Demmel (UCB))
(8:45 - 10:45am)

Outline

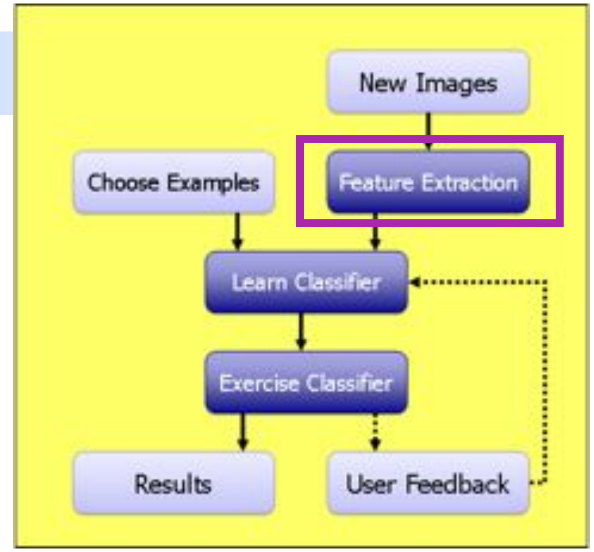
- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns

CBIR Application Framework

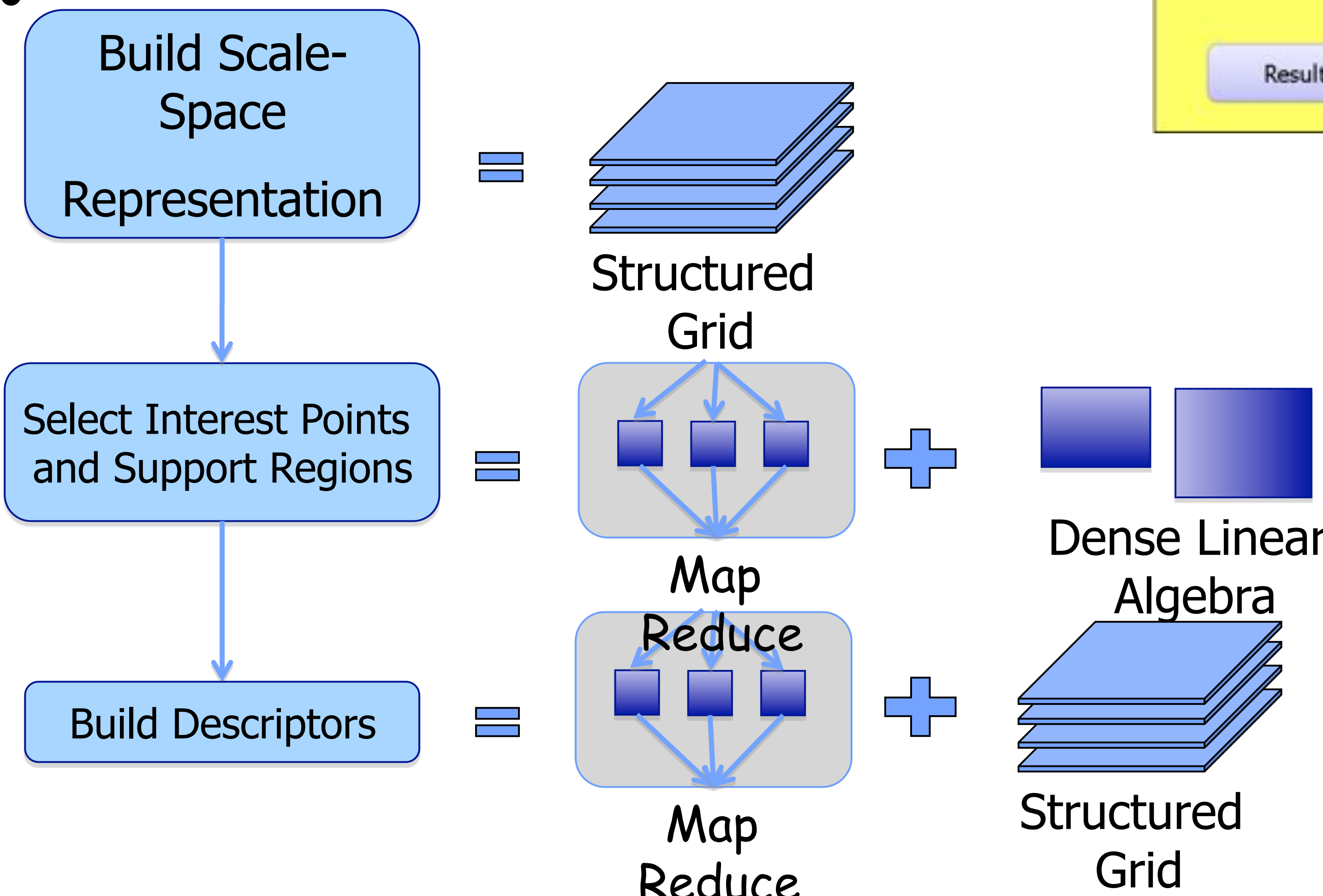


Catanzaro, Sundaram, Keutzer, “Fast SVM Training and Classification on Graphics Processors”, ICML 2008

Feature Extraction

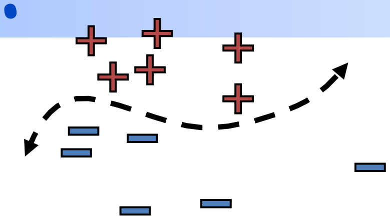


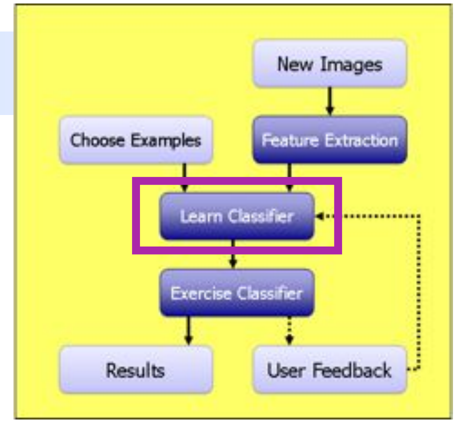
- Image is reduced to a set of low-dimensional feature vectors
-

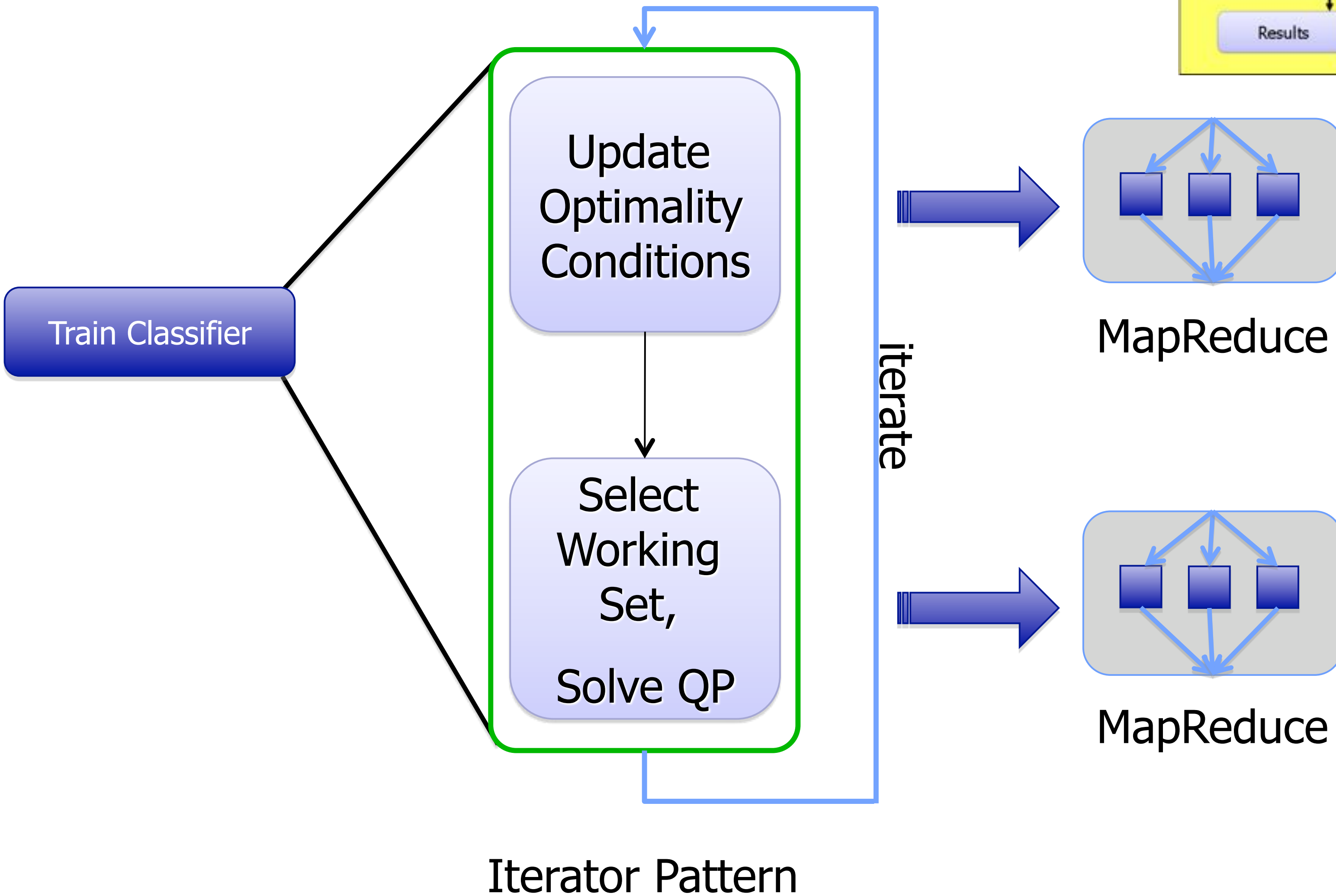


"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09

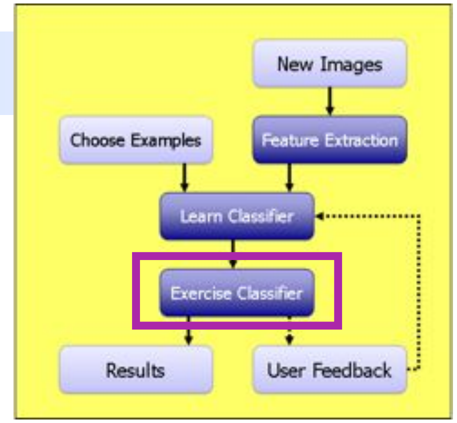
Train Classifier: SVM Training

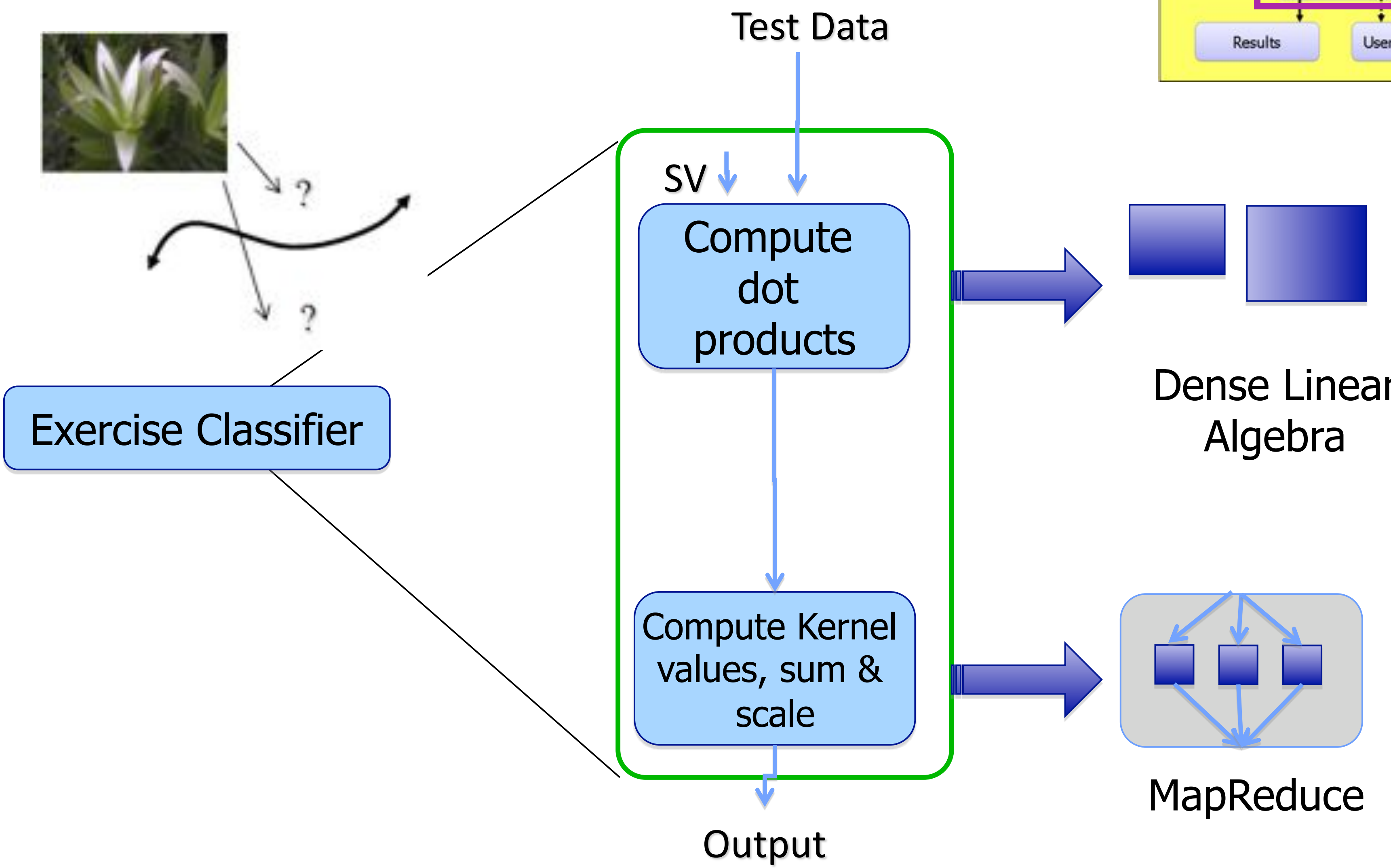




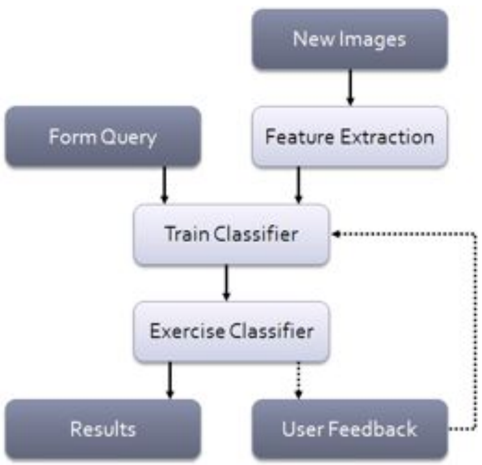


Exercise Classifier : SVM Classification





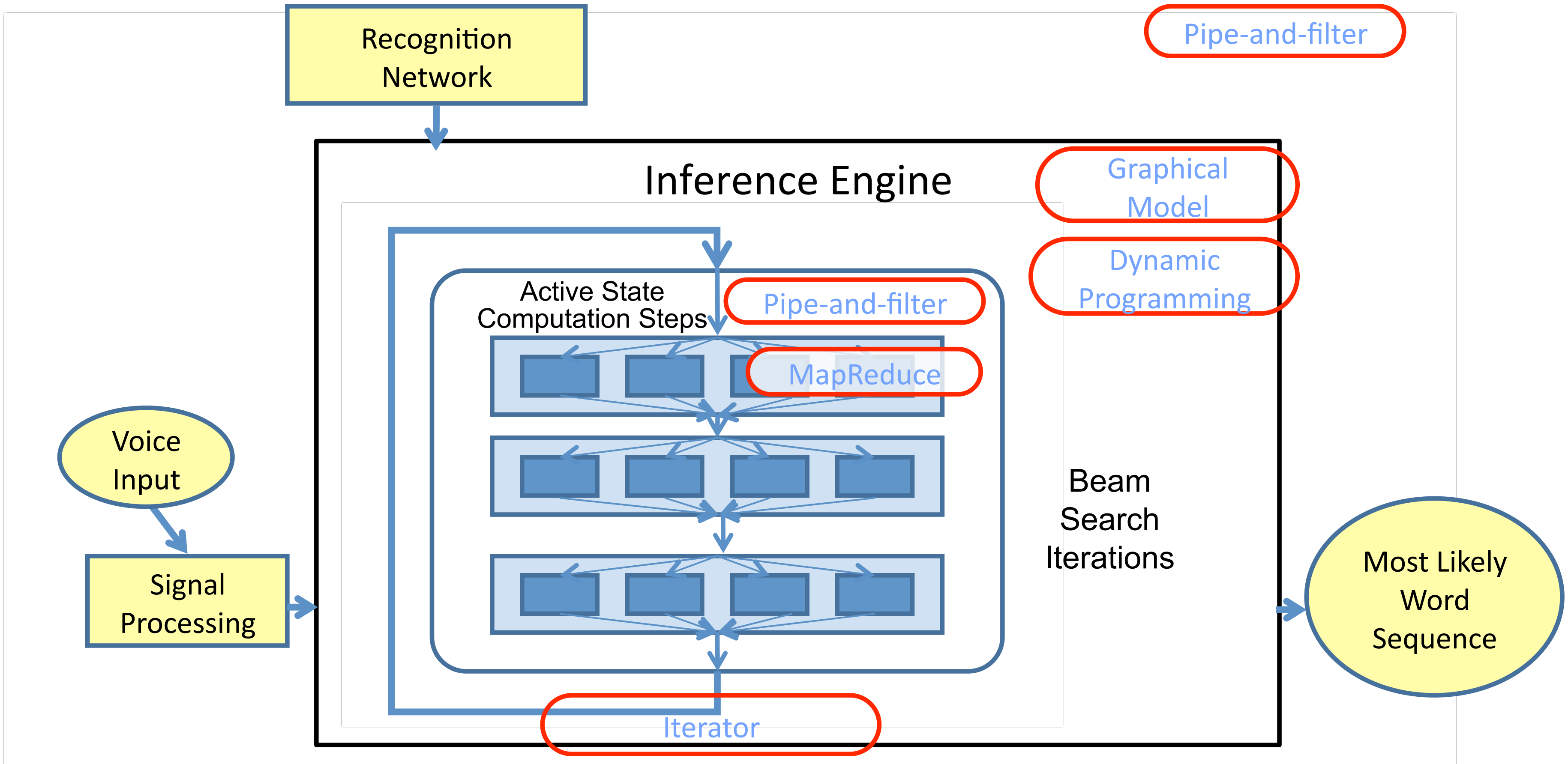
Support-Vector Machine Mini-Framework



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

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

Architecting Speech Recognition

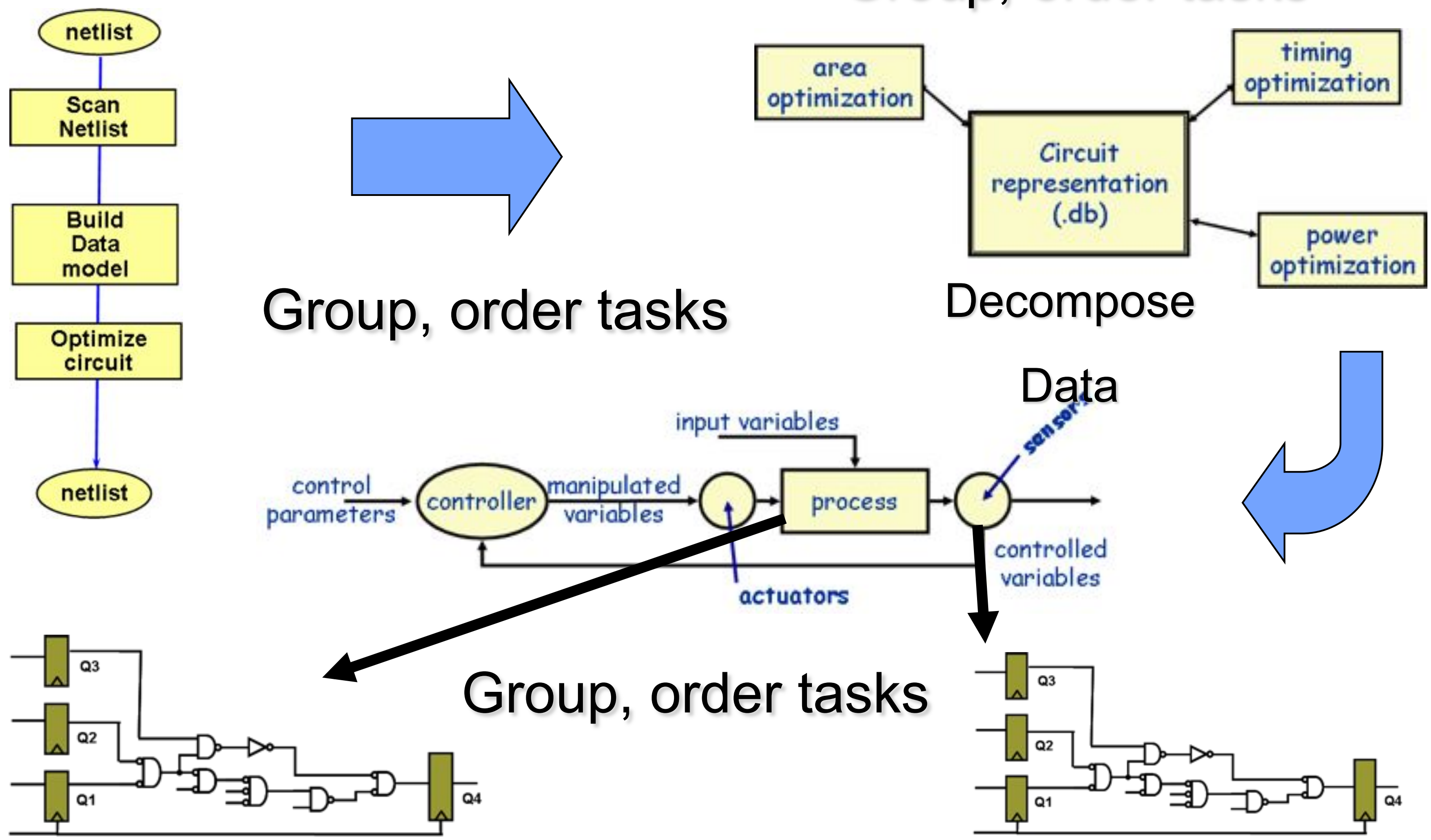


Large Vocabulary Continuous Speech Recognition Poster: Chong, Yi

Work also to appear at Emerging Applications for Manycore Architecture

Architecture of Logic Optimization

Group, order tasks



Graph algorithm pattern

Graph algorithm pattern

The take away

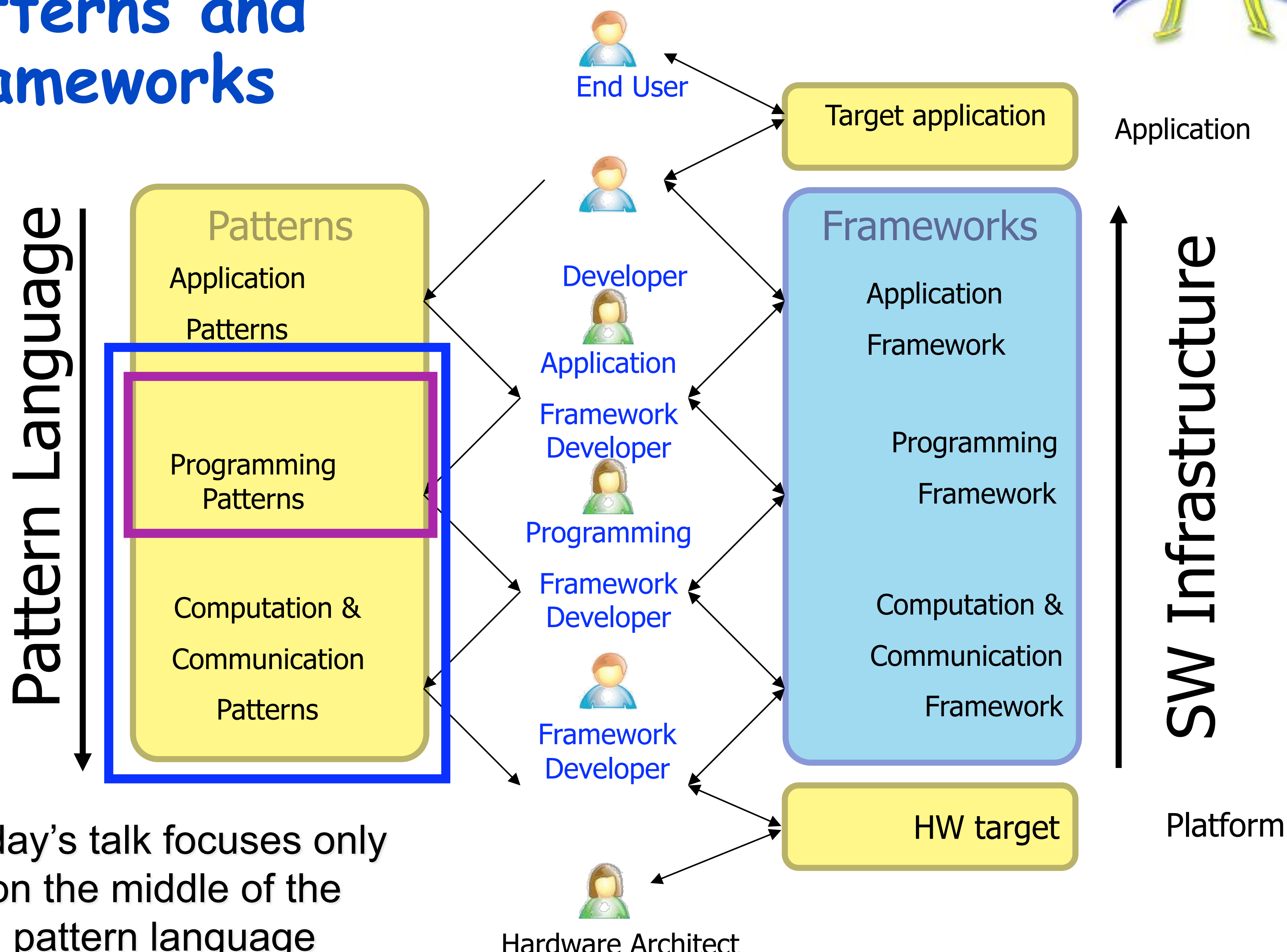
- My own experience has shown that a sound software architecture is the greatest single indicator of a software project's success.
- Software must be architected to achieve productivity, efficiency, and correctness
- SW architecture >> programming environments
  - >> programming languages
  - >> compilers and debuggers
  - (>>hardware architecture)
- Key to architecture (software or otherwise) is design patterns and a pattern language
- At the highest level our pattern language has:
  - Eight structural patterns
  - Thirteen computational patterns
- Yes, we really believe arbitrarily complex parallel software can built just from these!

What's next …

- Friday: Computational Patterns of Parallel Programming (James Demmel (UCB)) (8:45 - 10:45am)

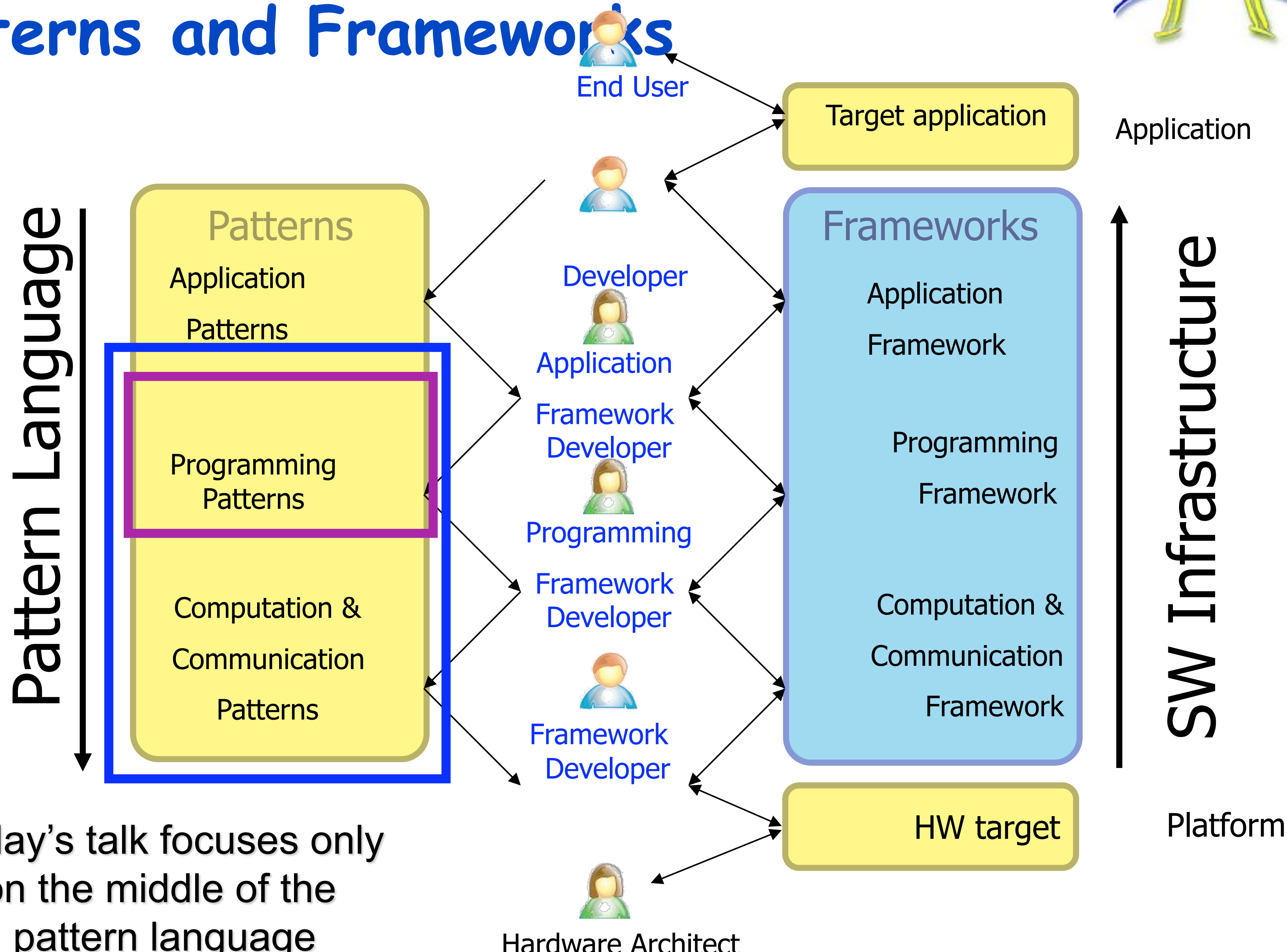
Extras

Patterns and Frameworks



Today's talk focuses only on the middle of the pattern language

Patterns and Frameworks



Today's talk focuses only on the middle of the pattern language

Elements of a Pattern - 1

- Name
  - It must have a meaningful name useful to remember the pattern and when it is used.
- Problem
  - A statement of the problem … a one-line preamble and the problem stated as a question.
- Context
  - The conditions under which the problem occurs. Defines when the pattern is applicable and the configuration of the system before the pattern is applied.
- Forces
  - A description of the relevant *forces* and constraints and how they interact/conflict with one another and with goals we wish to achieve. Defines the tension that characterizes a problem.

Elements of a Pattern - 2

- Solution
  - Instructions used to solve the problem. When done right, it resolves the tension defined in the forces section; flowing from the context and forces. We also define the new context for the system following application of the pattern.
- Invariant
  - What must be invariantly true for this pattern to work. May be stated in the form of Precondition, Invariant, Post-condition
- Examples
  - Examples to help the reader understand the pattern.
- Known Uses and frameworks
  - Cases where the pattern was used; preferably with literature references.
- Related Patterns
  - How does this pattern fit-in or work-with the other patterns in the pattern language.

*PLPP is the pattern language from Mattson, Sanders and Massingill

Definitions - 1

- Design Patterns: "Each design 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
- Structural patterns: design patterns that provide solutions to problems associated with the development of program structure
- Computational patterns: design patterns that provide solutions to recurrent computational problems

Definitions - 2

- Library: The software implementation of a computational pattern (e.g. BLAS) or a particular sub-problem (e.g. matrix multiply)
- Framework: An extensible software environment (e.g. Ruby on Rails) organized around a structural pattern (e.g. model-view-controller) that allows for programmer customization *only* in harmony with the structural pattern
- Domain specific language: A programming language (e.g. Matlab) that provides language constructs that particularly support a particular application domain. The language may also supply library support for common computations in that domain (e.g. BLAS). If the language is restricted to maintain fidelity to a structure and provides library support for common computations then it encompasses a framework (e.g. NPClick).

Eventually

1	Domain Experts	+	Application patterns & frameworks	→	End-user, application programs
2	Domain literate programming gurus (1% of the population).	+	Parallel patterns & programming frameworks	→	Application frameworks
3	Parallel programming gurus (1-10% of programmers)			→	Parallel programming frameworks

The hope is for Domain Experts to create parallel code with little or no understanding of parallel programming.

Leave hardcore "bare metal" efficiency layer programming to the parallel programming experts

Today

1 Domain Experts + Application patterns & frameworks

2 Domain literate programming gurus (1% of the population). + Parallel patterns & programming frameworks

3 Parallel programming gurus (1-10% of programmers)

→ End-user, application programs

→ Application frameworks

→ Parallel programming frameworks

- For the foreseeable future, domain experts, application framework builders, and parallel programming gurus will all need to learn the entire stack.
- That's why you all need to be here today!

People, Patterns, and Frameworks

	  Design Patterns	  Frameworks
Application Developer 	Uses application design patterns (e.g. feature extraction) to architect the application	Uses application frameworks (e.g. CBIR) to develop application
Application-Framework Developer 	Uses programming design patterns (e.g. Map/Reduce) to architect the application framework	Uses programming design patterns (e.g MapReduce) to develop the programming framework