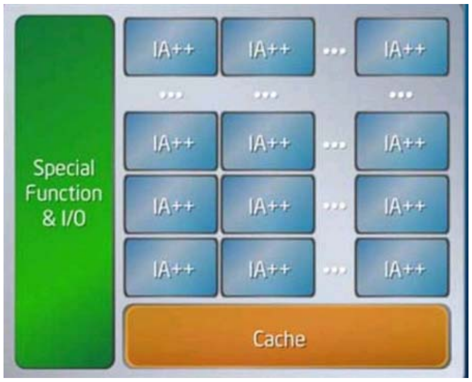
Architecting Parallel Software with Patterns

Kurt Keutzer, EECS, Berkeley
Tim Mattson, Intel
and the PALLAS team:
Michael Anderson, Bryan Catanzaro, (Jike Chong), Chao-Yue Lai,
Ekaterina Gonina, (Dorothea Kolossa), Mark Murphy,
David Sheffield, Bor-Yiing Su, Naryanan Sundaram,

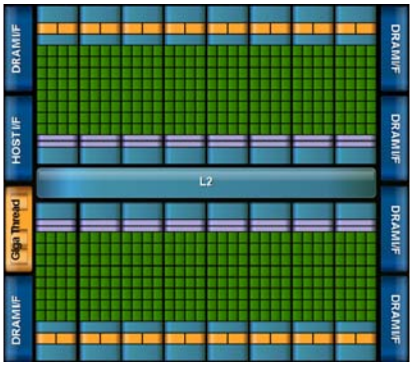
The Challenge of Parallelism

Intel: Larrabee



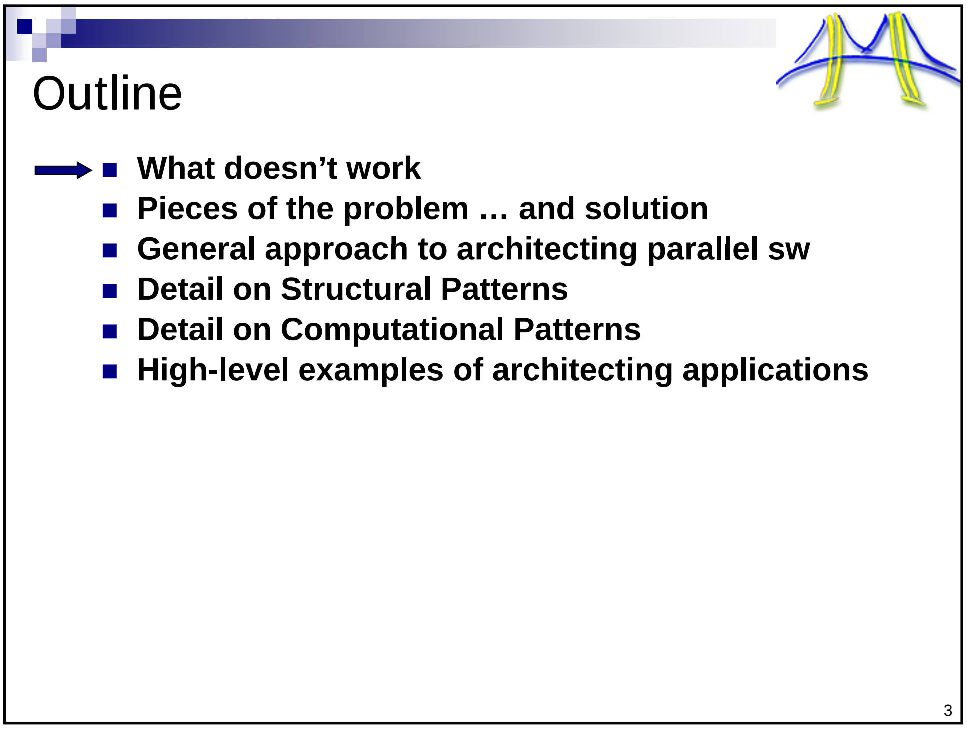
32 processors
each 16-wide vector unit

Nvidia: Fermi



16 processors
each 32-wide vector unit

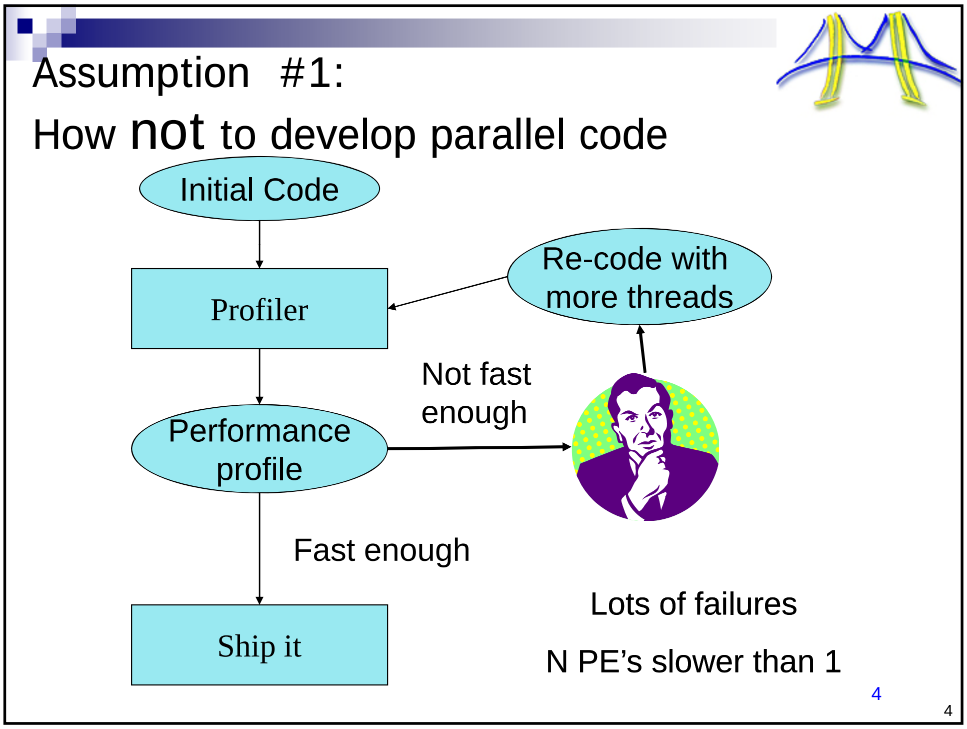
Programming highly parallel processors is the software challenge of our era



Outline

- What doesn’t work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

3



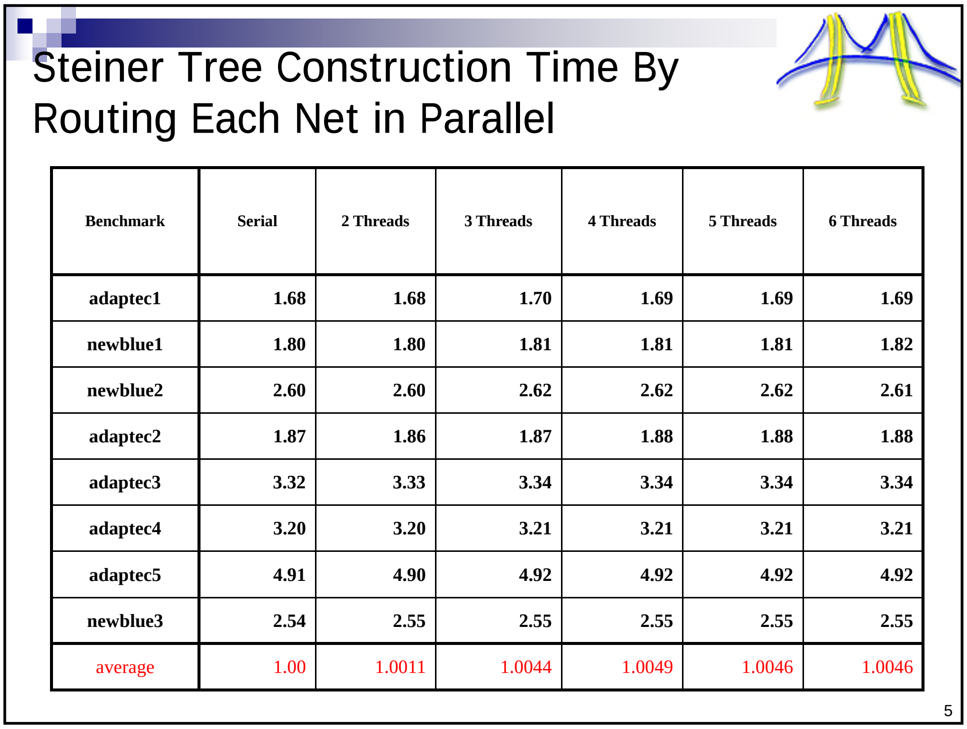
Assumption #1: How **not** to develop parallel code

Lots of failures

N PE’s slower than 1

4

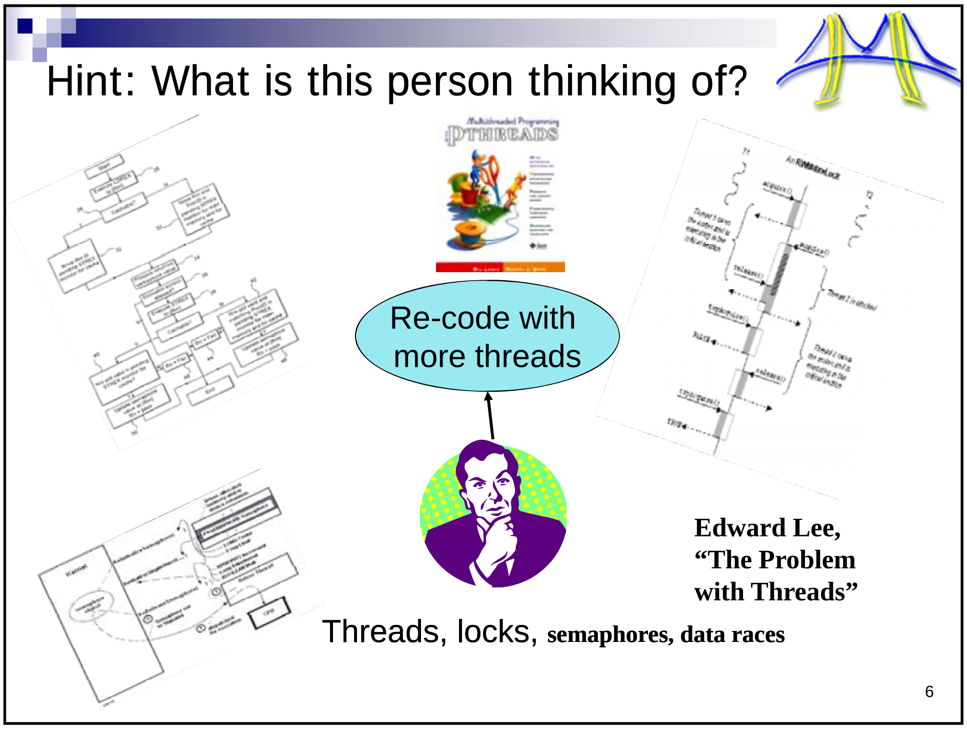
4



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

5



Hint: What is this person thinking of?

Re-code with more threads

Edward Lee, "The Problem with Threads"

Threads, locks, **semaphores, data races**

6

Outline

- What doesn't work
- → Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

7

Building software: where we begin

Grady Booch
OO Guru

Can be built by one person
Requires
    Minimal modeling
    Simple process
    Simple tools

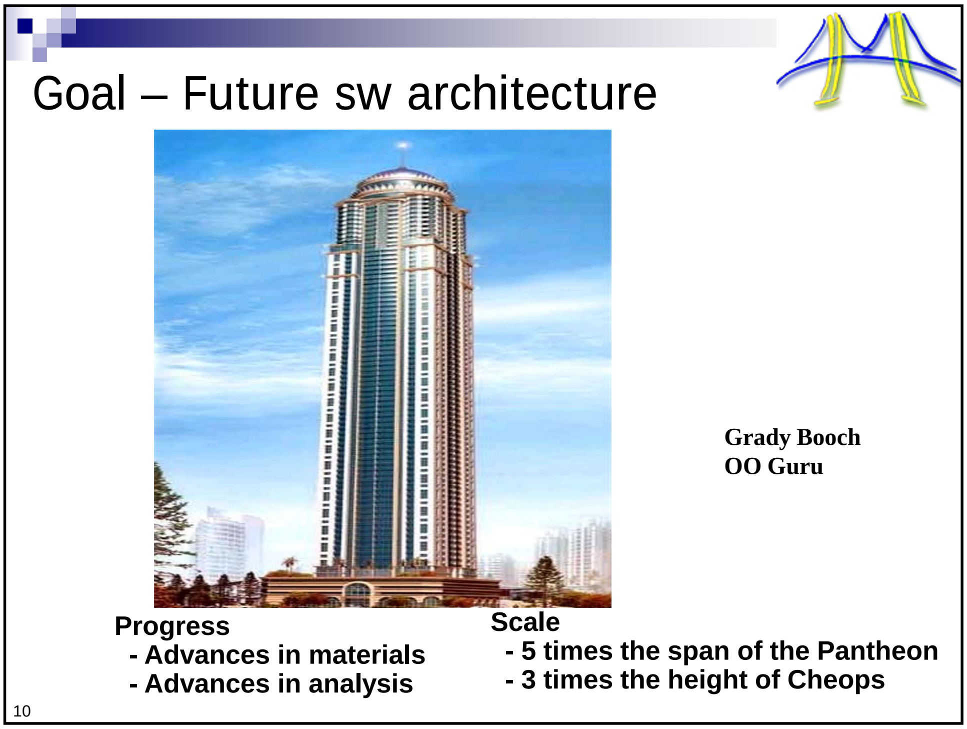
8

The progress of Object Oriented Programming

Built most efficiently and timely by a team
Requires
- Modeling
- Well-defined process
- Power tools

Grady Booch
OO Guru

9



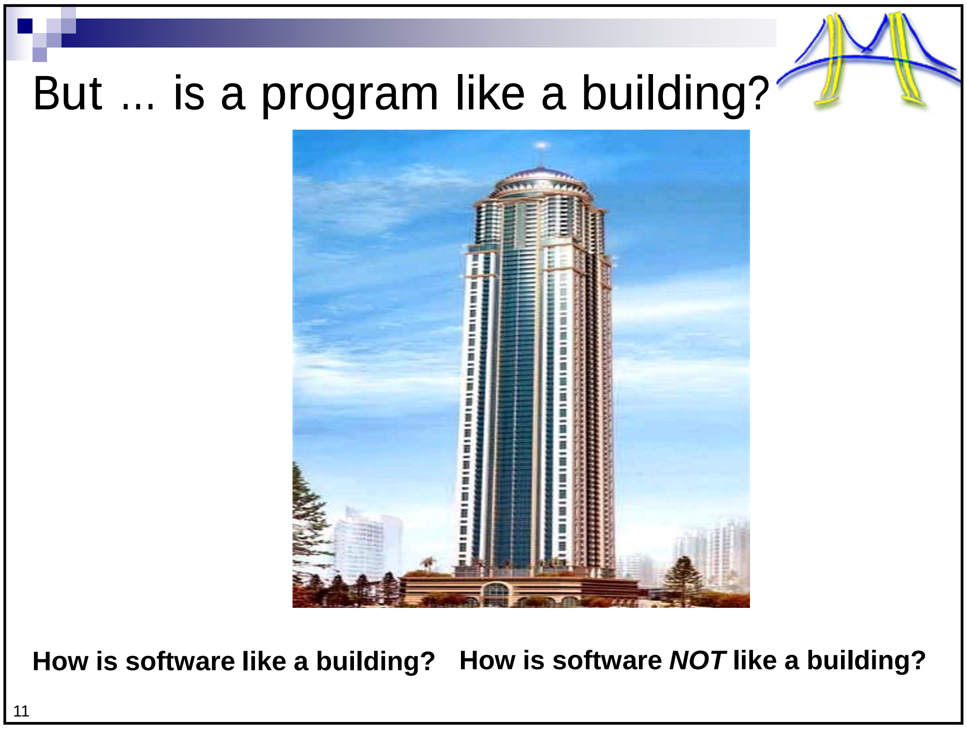
Goal – Future sw architecture

Grady Booch
OO Guru

Progress
- **Advances in materials**
- **Advances in analysis**

Scale
- **5 times the span of the Pantheon**
- **3 times the height of Cheops**

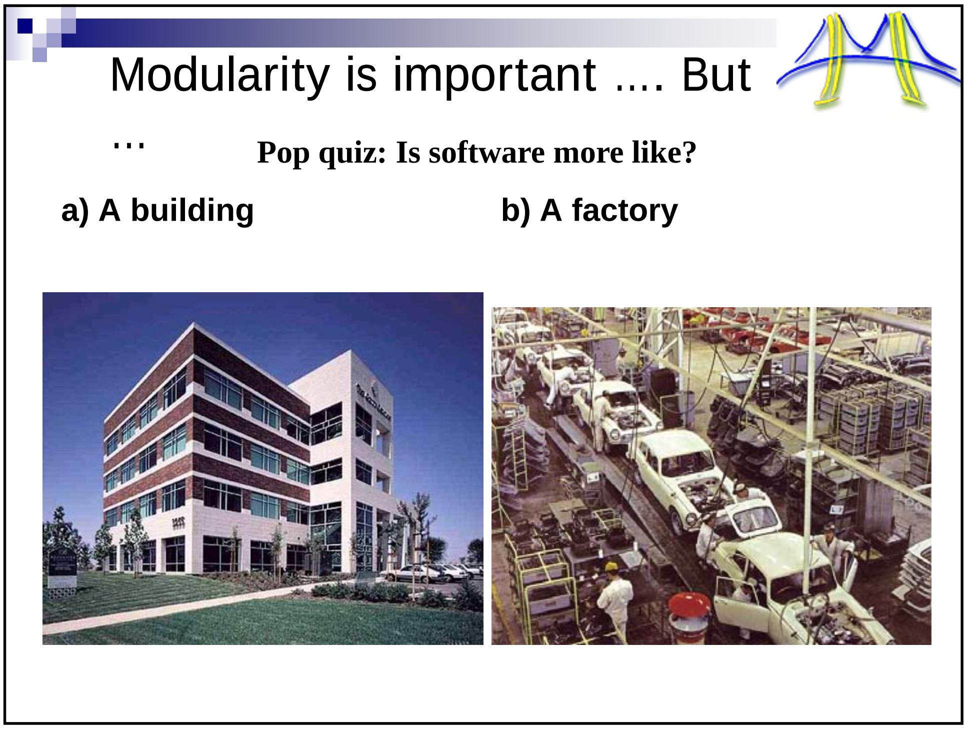
10



But ... is a program like a building?

How is software like a building? **How is software *NOT* like a building?**

11



Modularity is important …. But …

Pop quiz: Is software more like?

a) A building **b) A factory**

Object-Oriented Programming

Focused on:

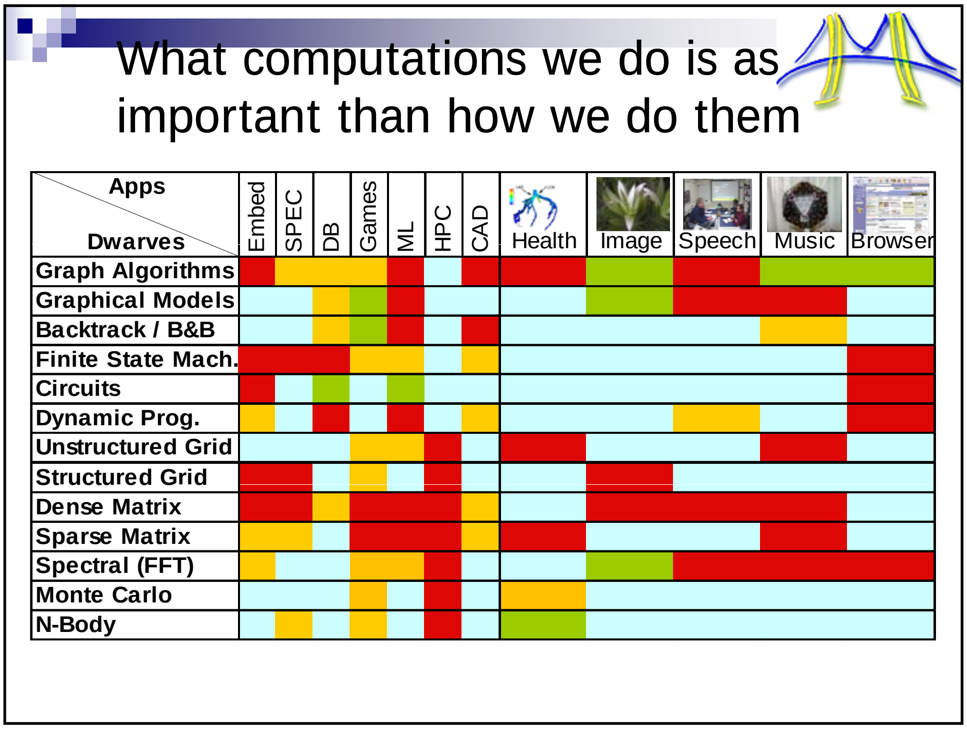
- **Program modularity**
- **Data locality**
- **Architectural styles**
- **Design patterns**

Neglected:

- **Application concurrency**
- **Computational details**
- **Parallel implementations**

Modularity and locality have proved to be essential concepts for:
• Design
•Implementation
•Verification/test

13



What computations we do is as important than how we do them

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	🟦	🟨	🟦	🟨	🟦	🟥	🟦	🟩	🟦	🟦	🟦	🟦

High performance computing

HPC knows a lot about application concurrency, efficient programming, and parallel implementation

$$x_c \leftarrow \sum_j g_{cj} * x_j$$

$$x \leftarrow WS_\lambda\{W^* x\}$$

$$x \leftarrow F(P^T y + P_c^T P_c F^* x)$$

$$\boxed{\frac{\partial V}{\partial t} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.}$$

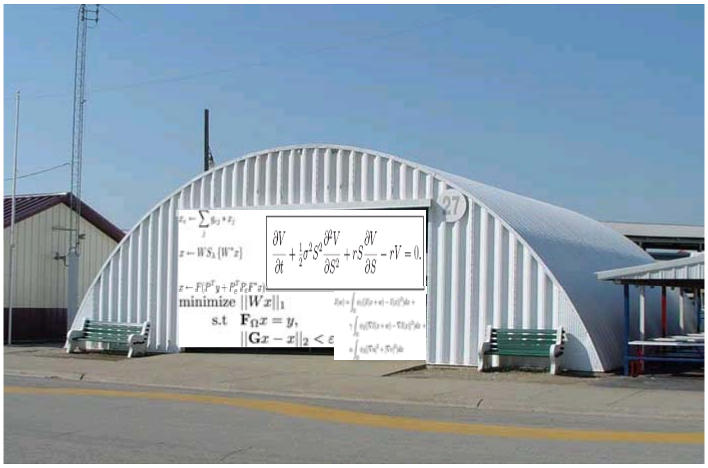
$$\begin{aligned} &\text{minimize } \|Wx\|_1 \\ &\quad \text{s.t } \mathbf{F}_\Omega x = y, \\ &\qquad \|\mathbf{G}x - x\|_2 < \varepsilon \end{aligned}$$

$$\begin{aligned} J(w) = &\int_\Omega \psi_1(|I(x+w) - I(x)|^2)dx + \\ \gamma &\int_\Omega \psi_2(|\nabla I(x+w) - \nabla I(x)|^2)dx + \\ \alpha &\int_\Omega \psi_3(|\nabla u|^2 + |\nabla v|^2)dx \end{aligned}$$

15

HPC approach to sw architecture

Technically this is known as a monolithic architecture

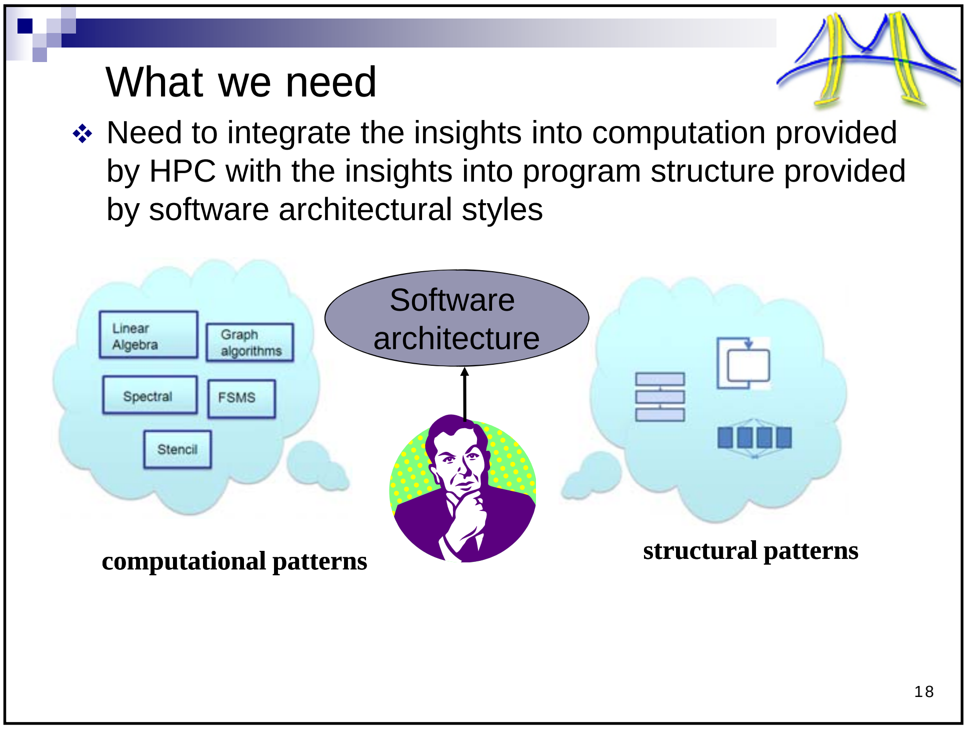


16

What’s a better metaphor for sw development?

© Kurt Keutzer

17



What we need

- Need to integrate the insights into computation provided by HPC with the insights into program structure provided by software architectural styles

Linear Algebra

Graph algorithms

Spectral

FSMS

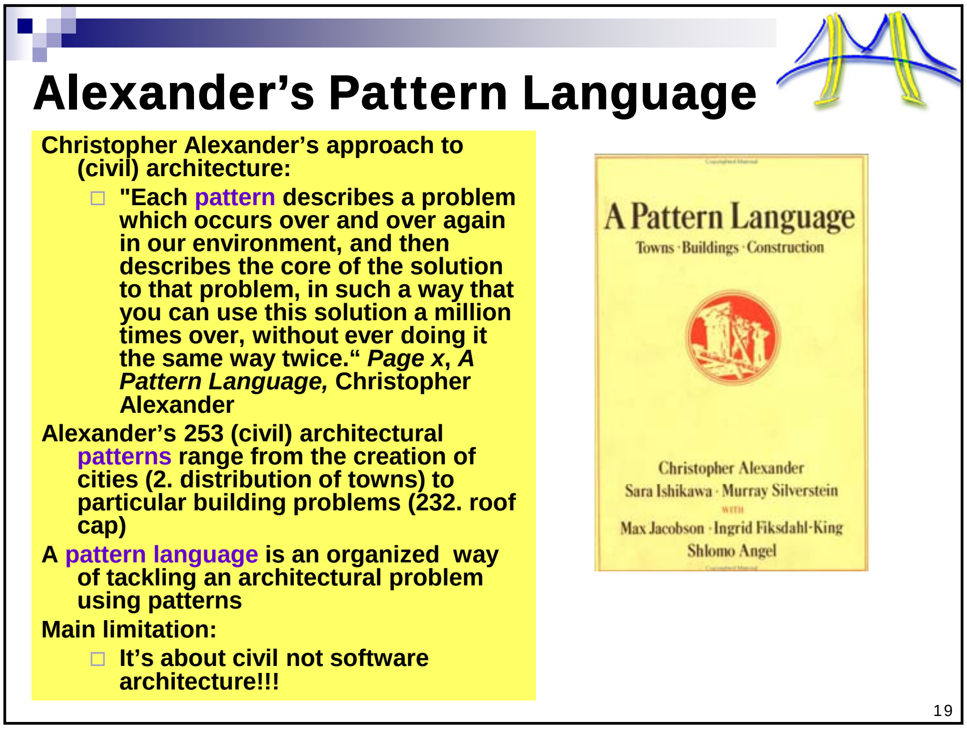
Stencil

Software architecture

computational patterns

structural patterns

18



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)

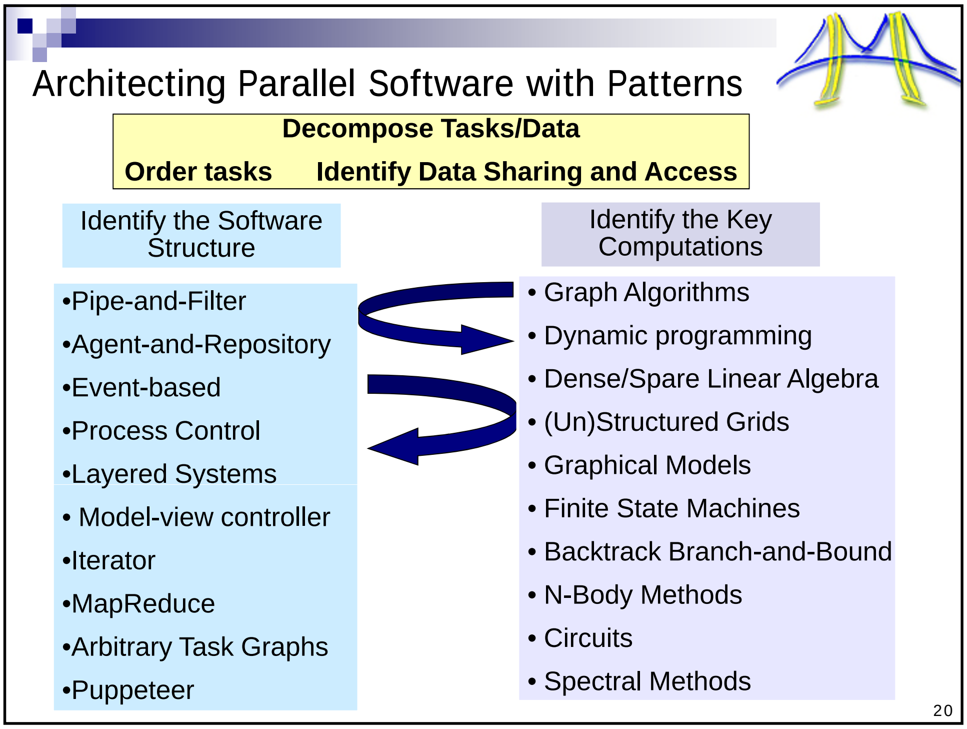
A pattern language is an organized way of tackling an architectural problem using patterns

Main limitation:

- □ It's about civil not software architecture!!!

A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

19



Architecting Parallel Software with Patterns

Decompose Tasks/Data

Order tasks **Identify Data Sharing and Access**

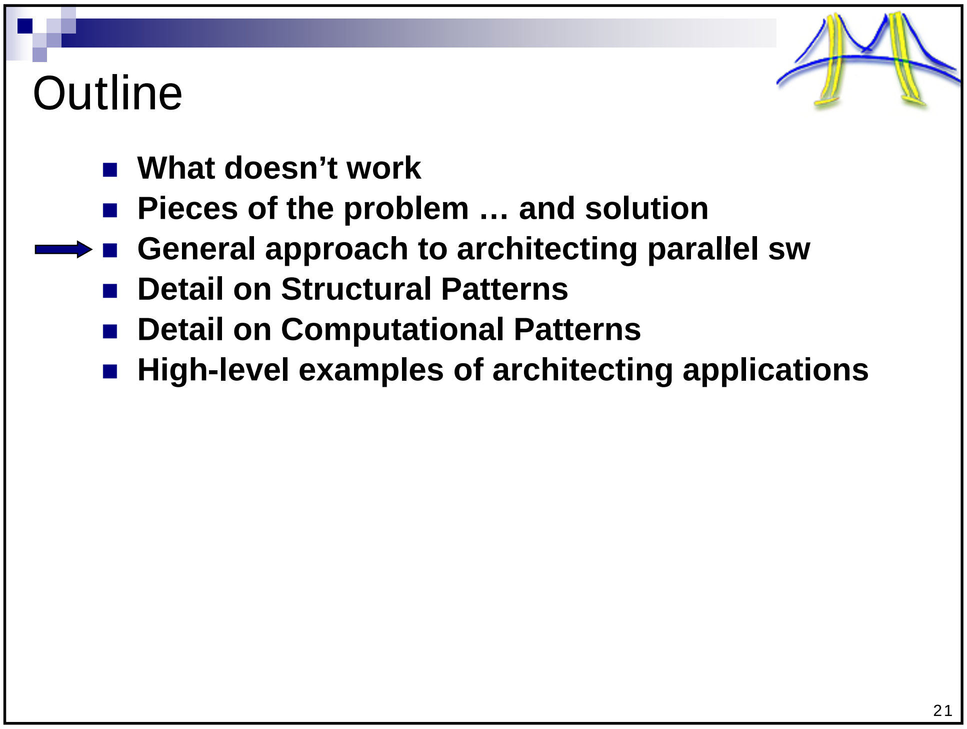
Identify the Software Structure

- •Pipe-and-Filter
- •Agent-and-Repository
- •Event-based
- •Process Control
- •Layered Systems
- • Model-view controller
- •Iterator
- •MapReduce
- •Arbitrary Task Graphs
- •Puppeteer

Identify the Key Computations

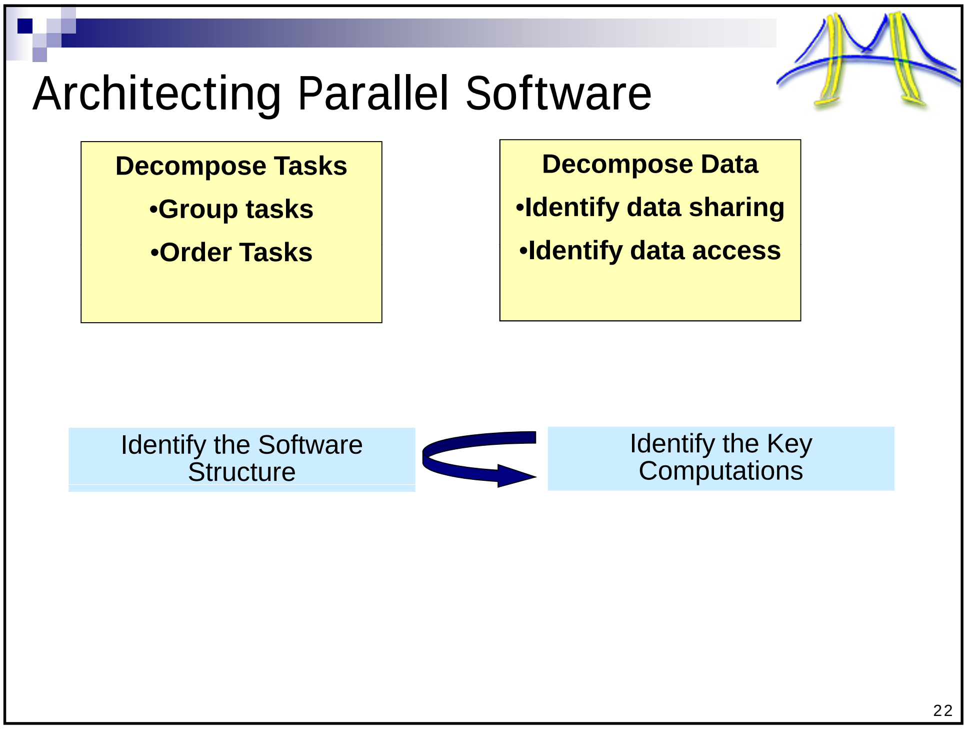
- • Graph Algorithms
- • Dynamic programming
- • Dense/Spare Linear Algebra
- • (Un)Structured Grids
- • Graphical Models
- • Finite State Machines
- • Backtrack Branch-and-Bound
- • N-Body Methods
- • Circuits
- • Spectral Methods

20



Outline

- What doesn’t work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications



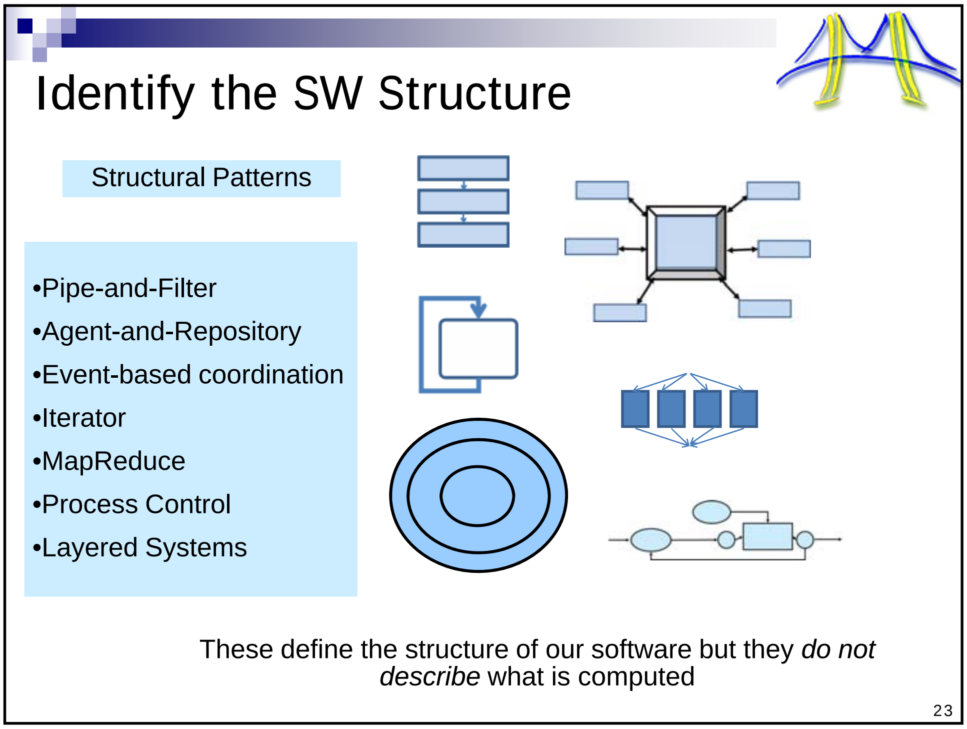
Architecting Parallel Software

Decompose Tasks
- •Group tasks
- •Order Tasks

Decompose Data
- •Identify data sharing
- •Identify data access

Identify the Software Structure

Identify the Key Computations



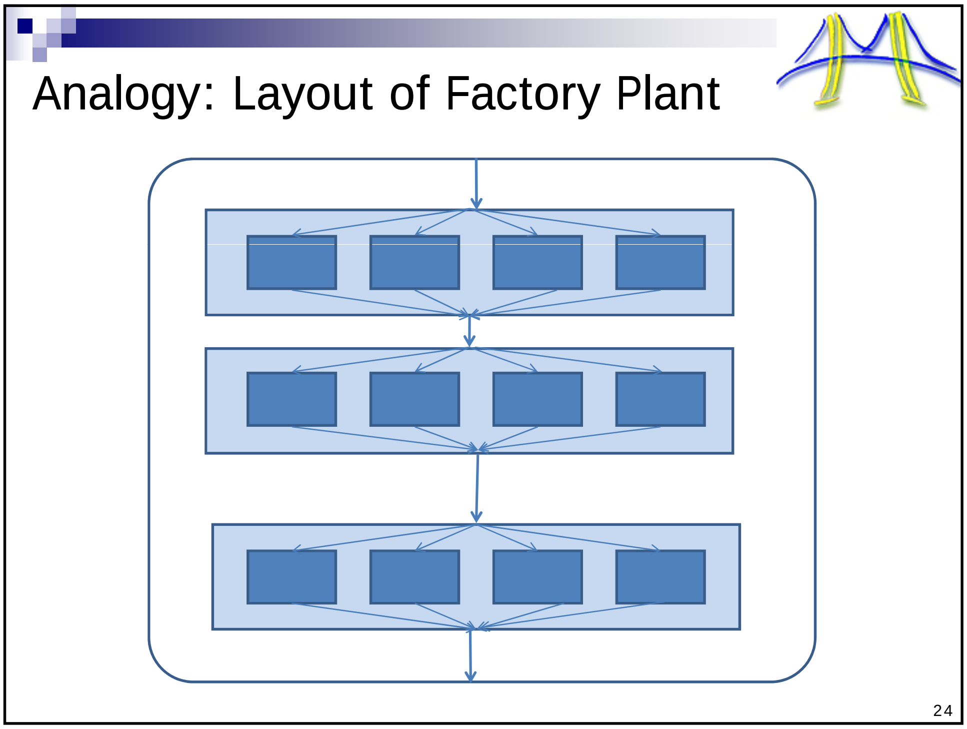
Identify the SW Structure

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

23



Analogy: Layout of Factory Plant

24

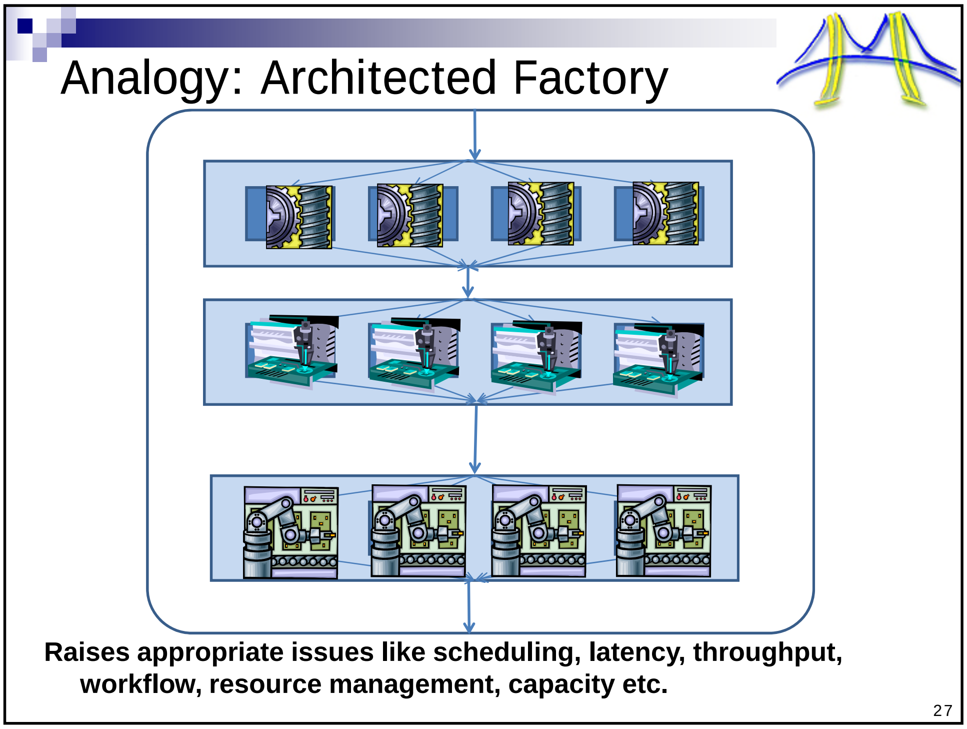
Identify Key Computations

Computational Patterns

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

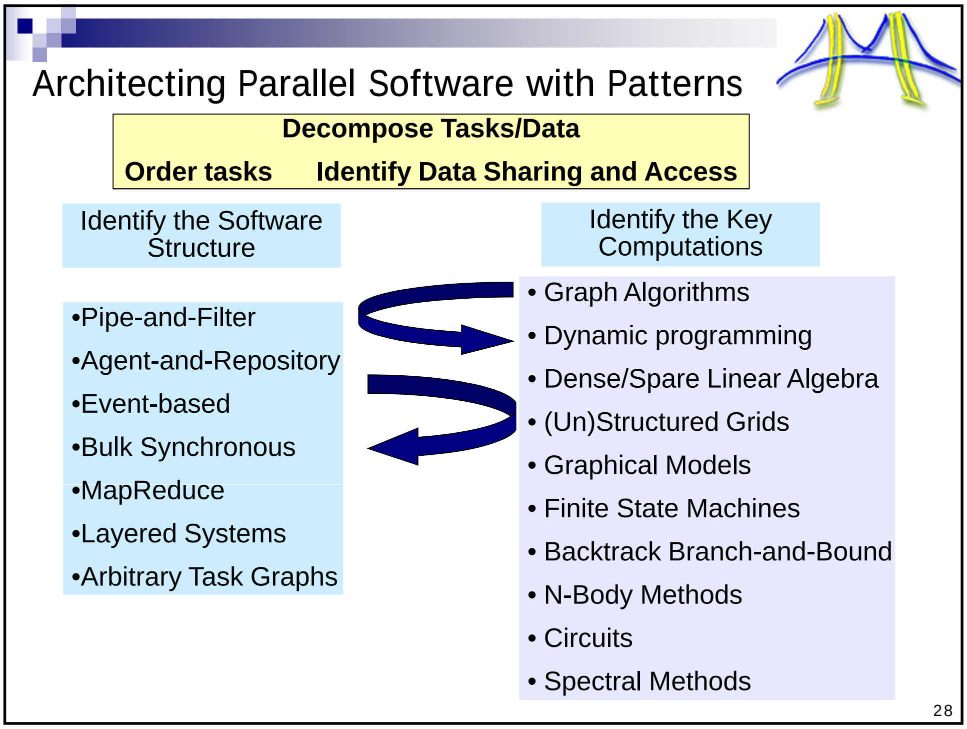
Computational patterns describe the key computations but not how they are implemented

Analogy: Machinery of the Factory



Analogy: Architected Factory

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



Architecting Parallel Software with Patterns

Decompose Tasks/Data

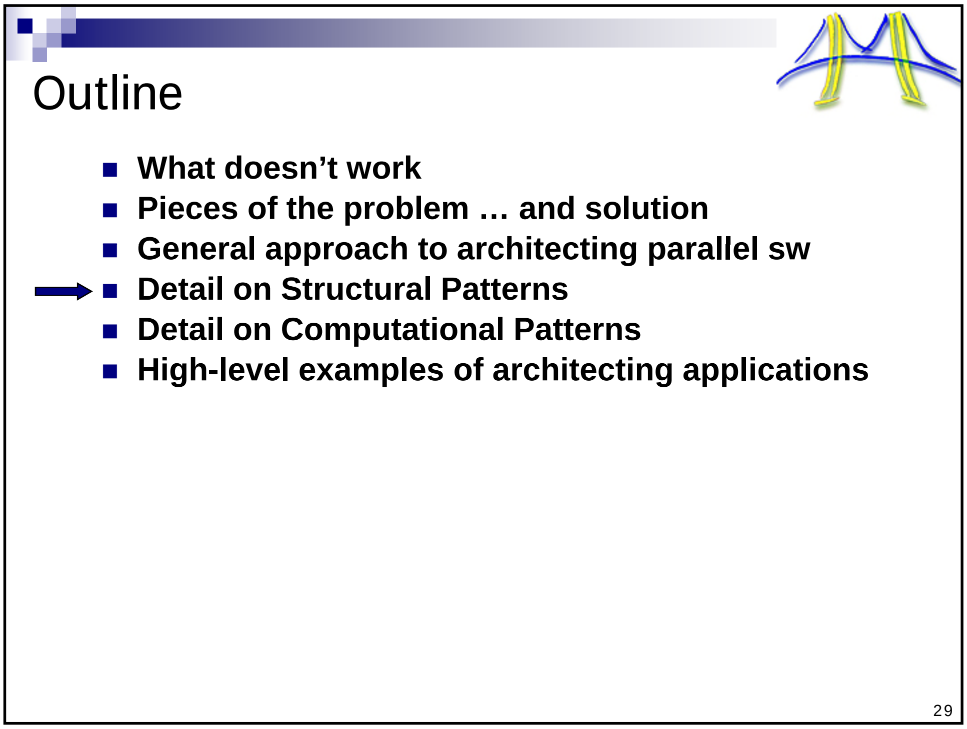
Order tasks **Identify Data Sharing and Access**

Identify the Software Structure

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

Identify the Key Computations

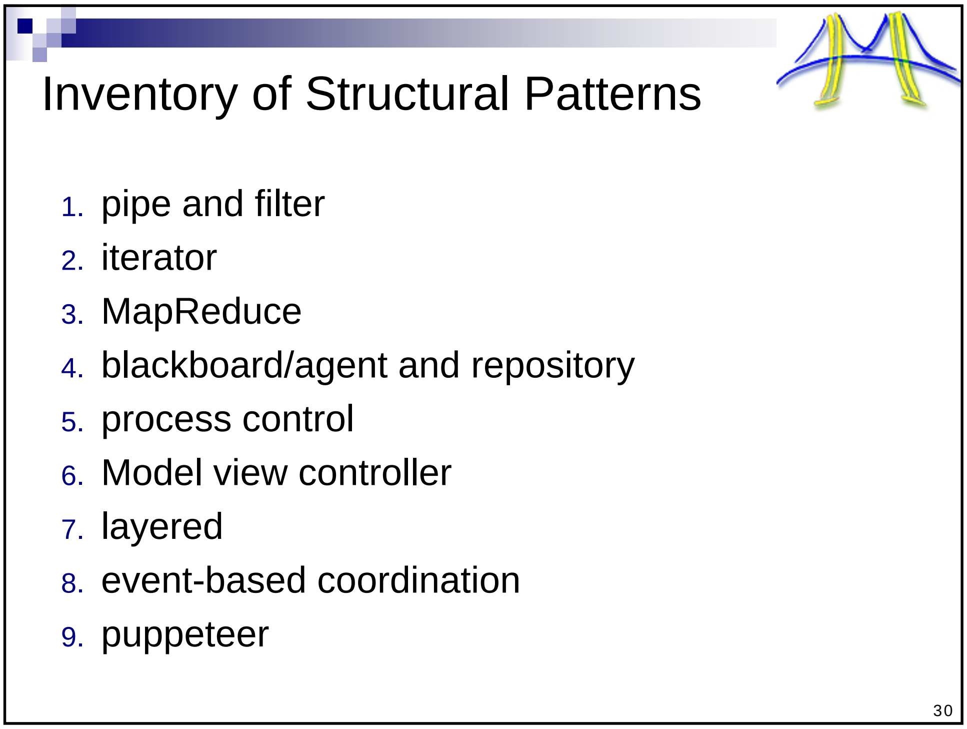
- Graph Algorithms
- Dynamic programming
- Dense/Spare Linear Algebra
- (Un)Structured Grids
- Graphical Models
- Finite State Machines
- Backtrack Branch-and-Bound
- N-Body Methods
- Circuits
- Spectral Methods



Outline

- **What doesn't work**
- **Pieces of the problem … and solution**
- **General approach to architecting parallel sw**
- → **Detail on Structural Patterns**
- **Detail on Computational Patterns**
- **High-level examples of architecting applications**

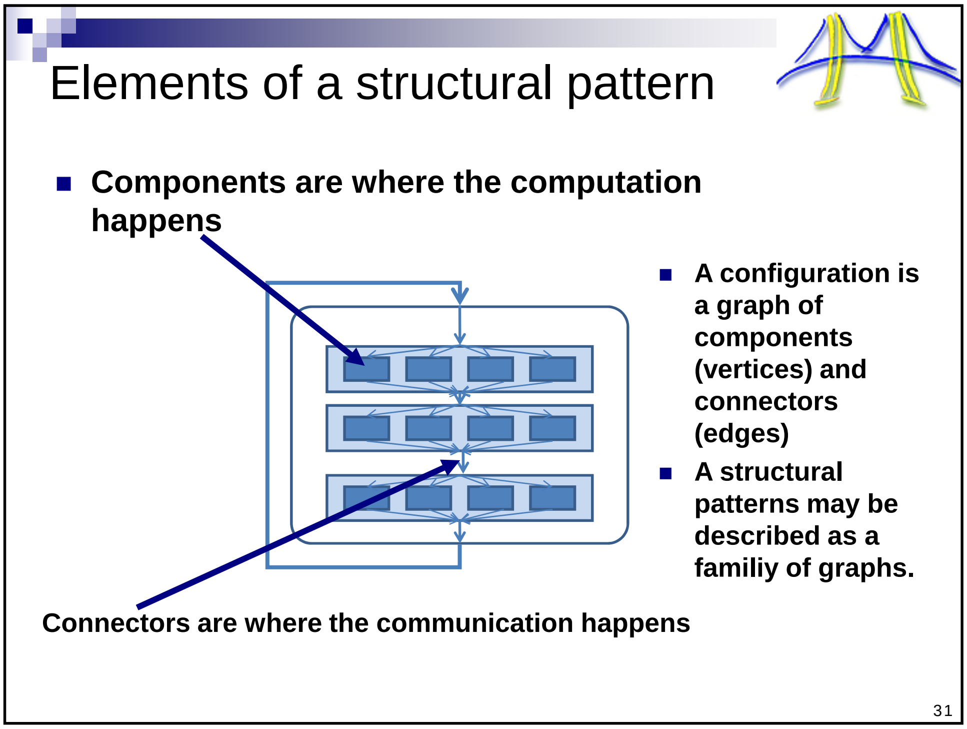
29



Inventory of Structural Patterns

1. pipe and filter
2. iterator
3. MapReduce
4. blackboard/agent and repository
5. process control
6. Model view controller
7. layered
8. event-based coordination
9. puppeteer

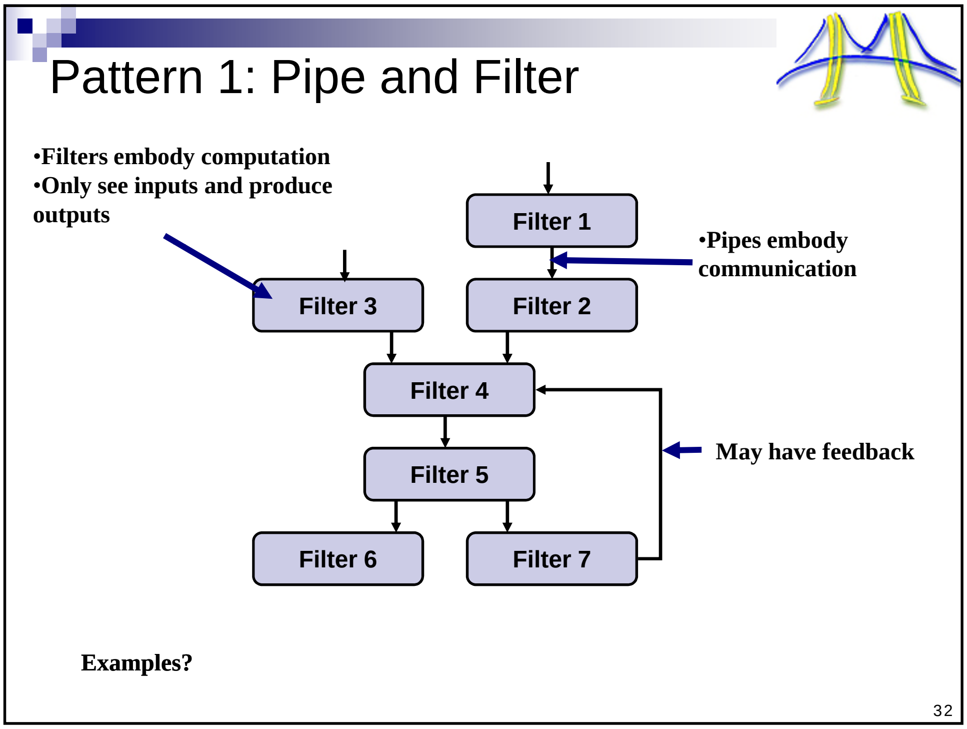
30



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



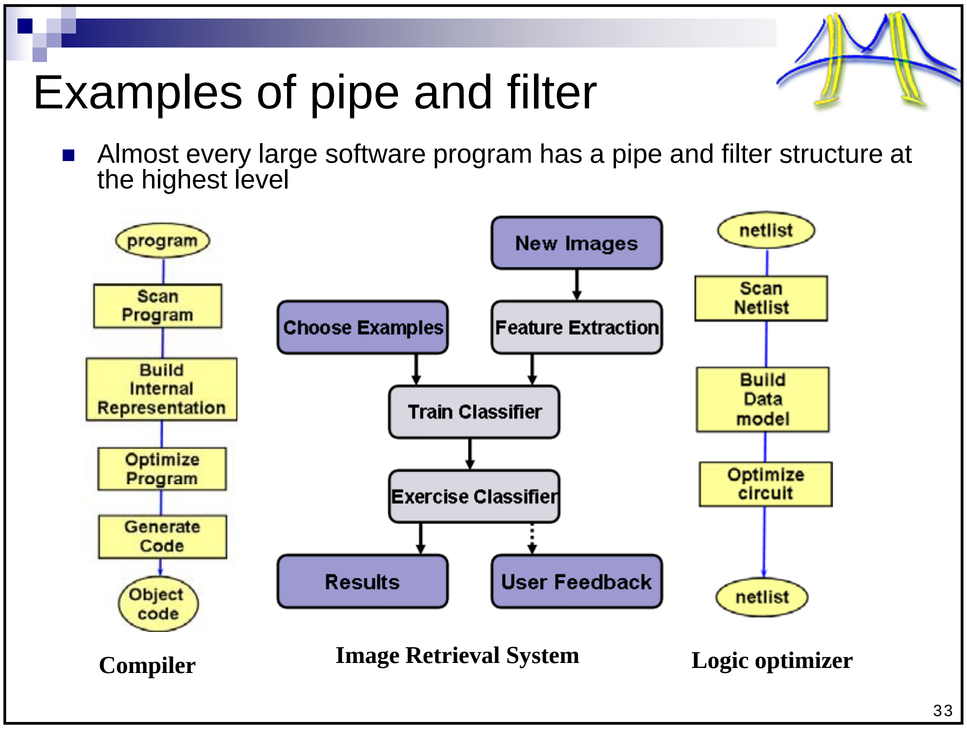
Pattern 1: Pipe and Filter

•Filters embody computation
•Only see inputs and produce outputs

•Pipes embody communication

May have feedback

Examples?



Examples of pipe and filter

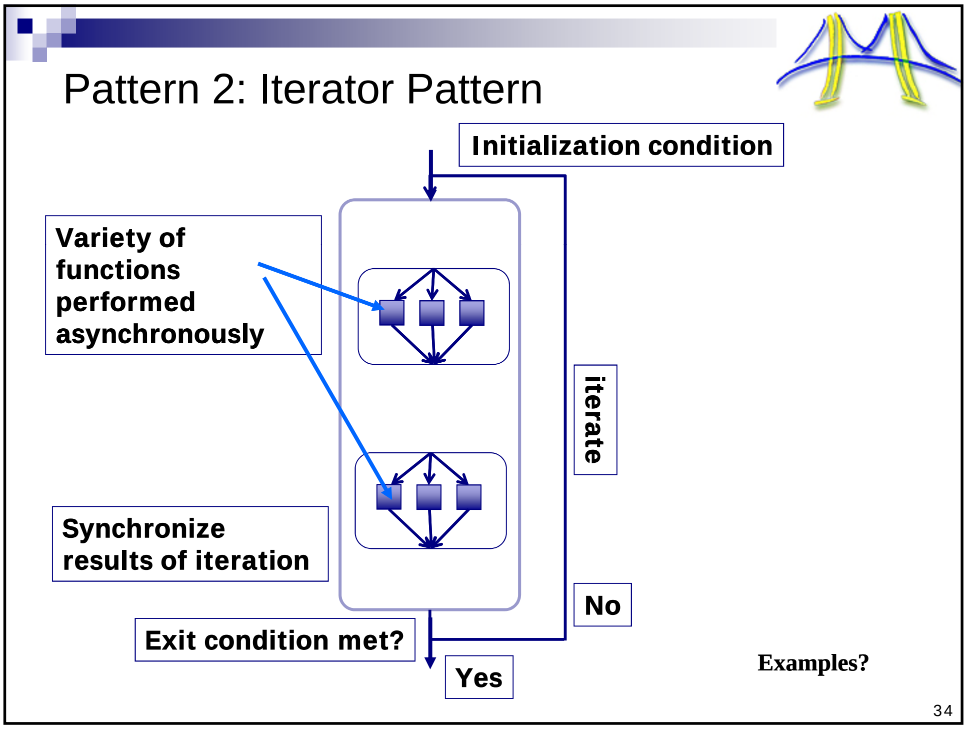
- Almost every large software program has a pipe and filter structure at the highest level

Compiler

Image Retrieval System

Logic optimizer

33

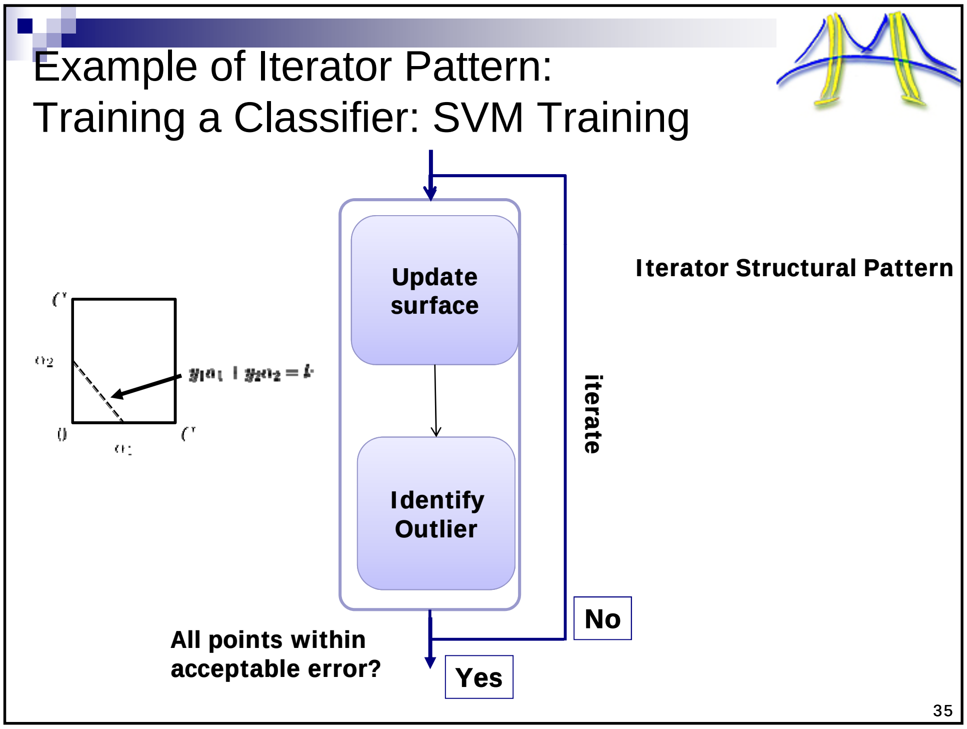


Pattern 2: Iterator Pattern

Initialization condition
Variety of functions performed asynchronously
iterate
Synchronize results of iteration
No
Exit condition met?
Yes

Examples?

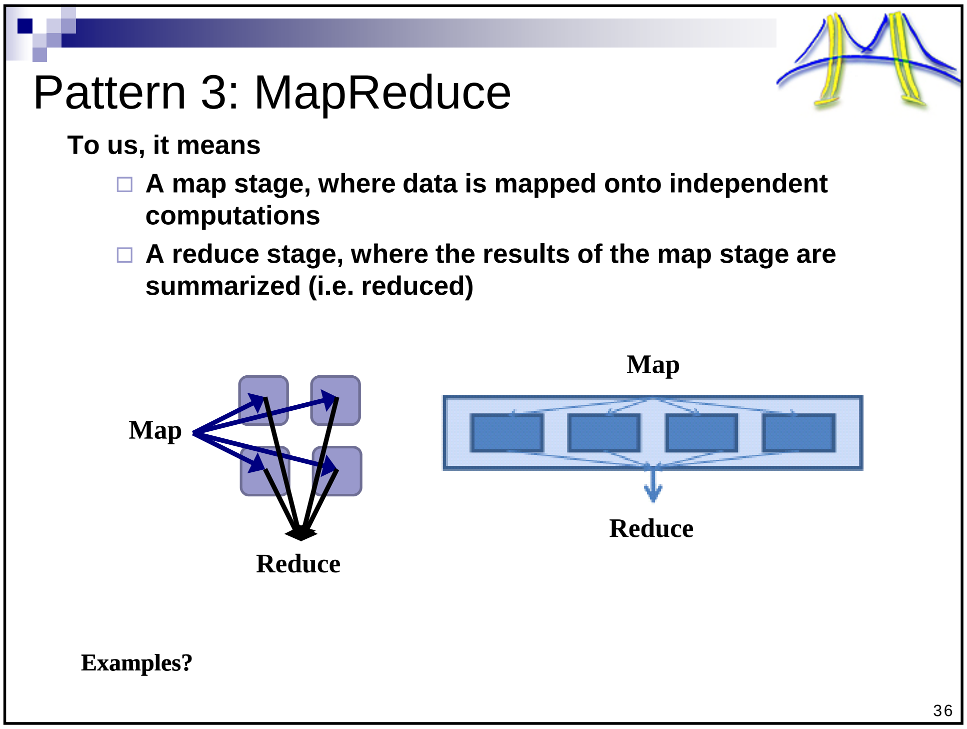
34



Example of Iterator Pattern: Training a Classifier: SVM Training

Iterator Structural Pattern

All points within acceptable error?

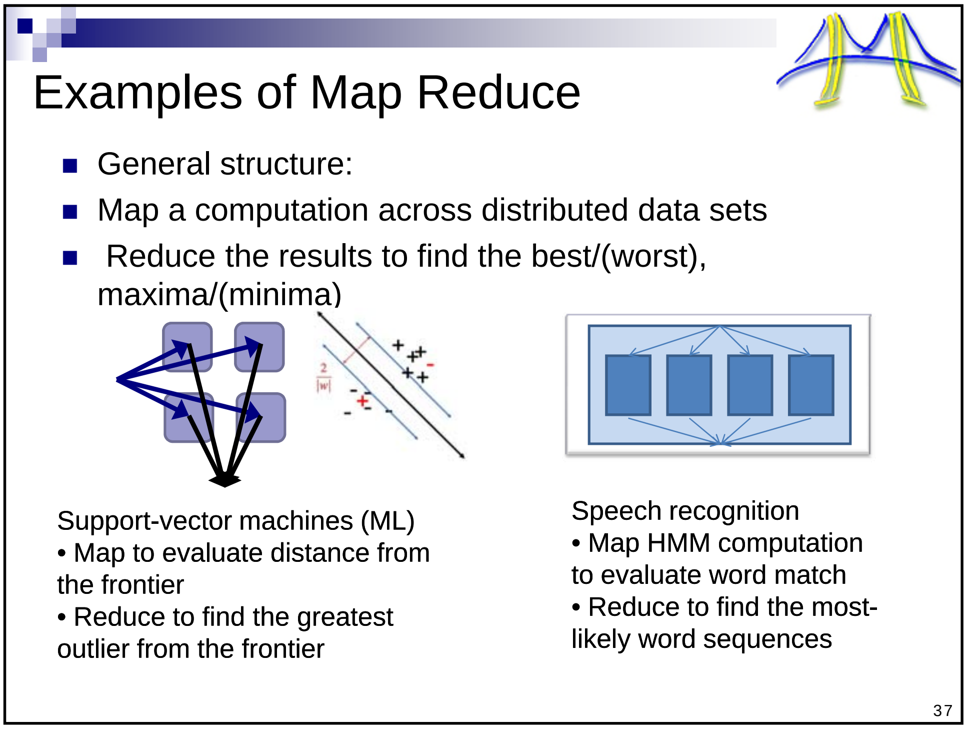


Pattern 3: MapReduce

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?

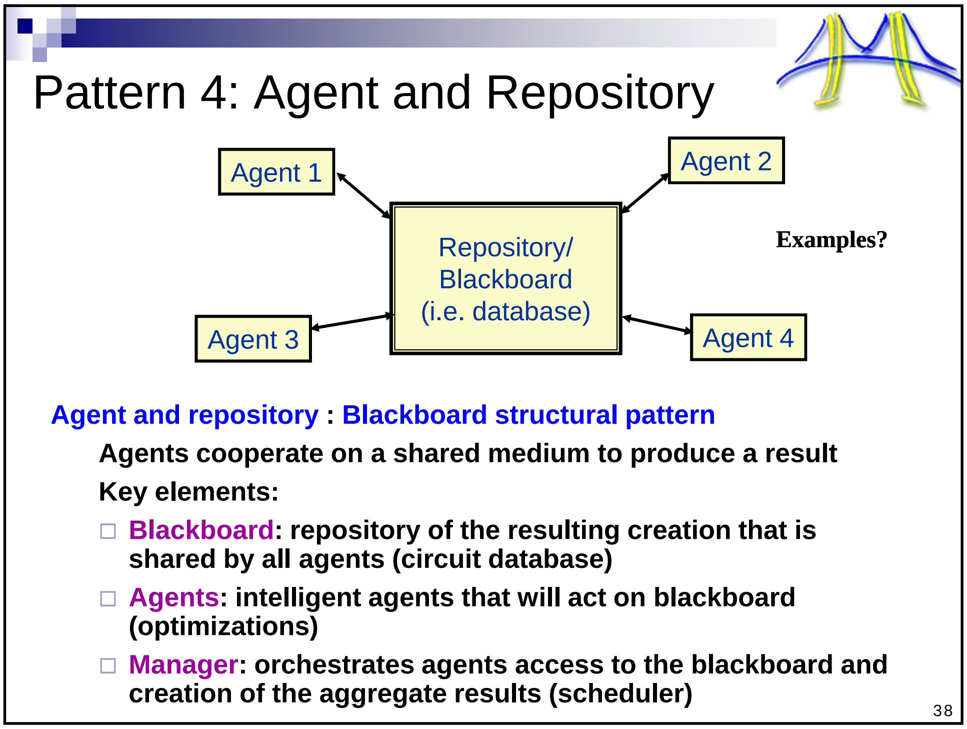


Examples of Map Reduce

- 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

Agent 1

Agent 2

Repository/
Blackboard
(i.e. database)

Agent 3

Agent 4

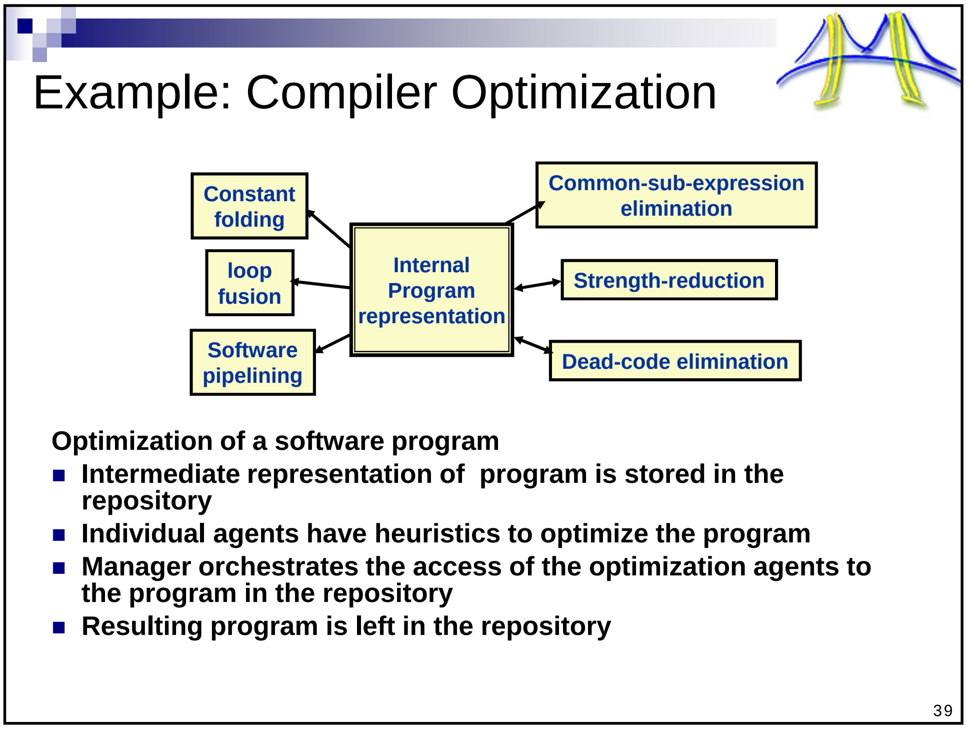
Examples?

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)**

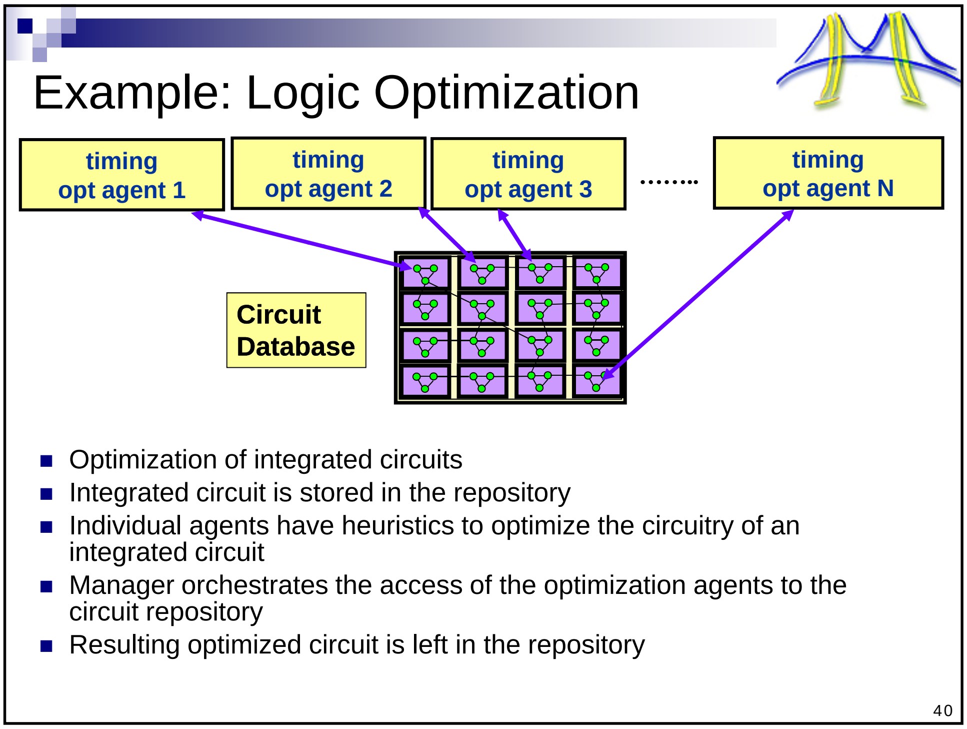


Example: Compiler Optimization

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**

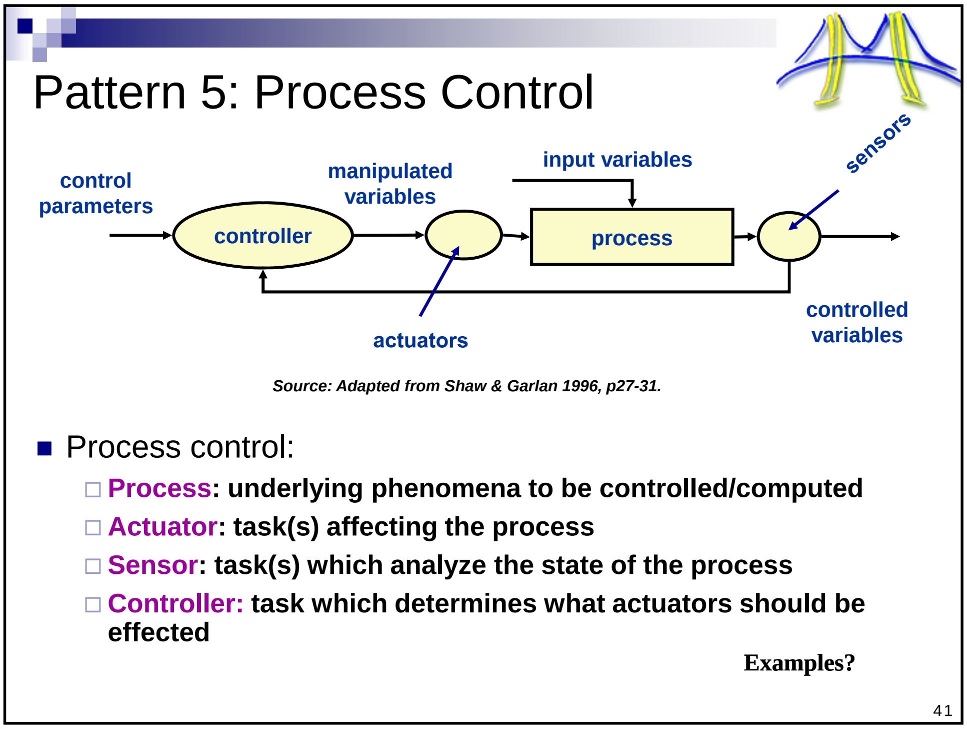
39



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

40



Pattern 5: Process Control

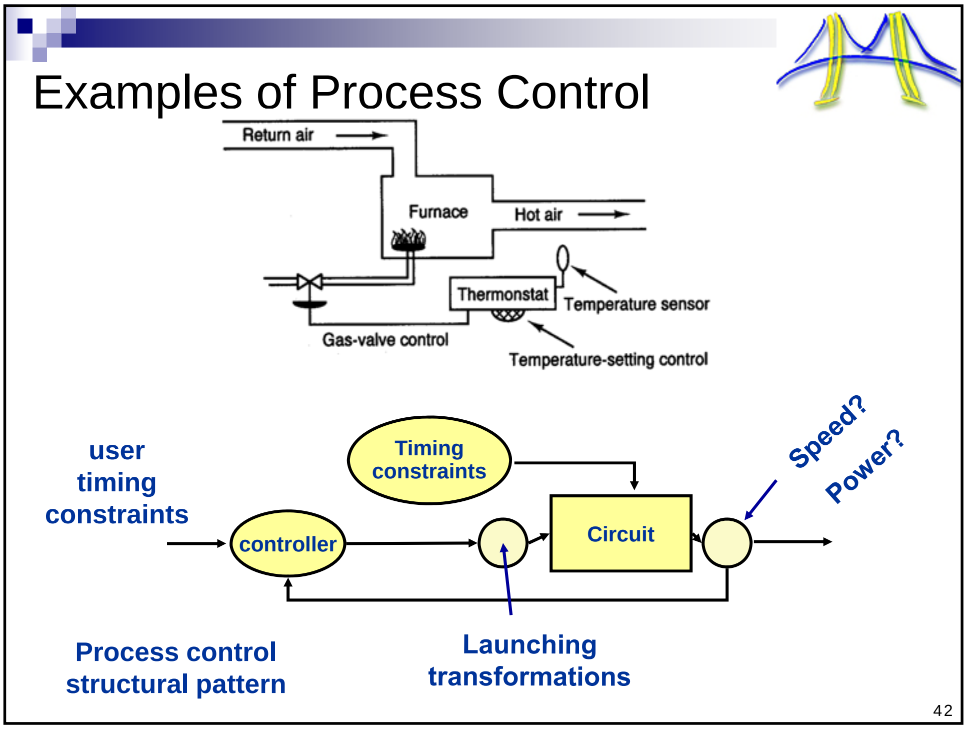
control parameters
controller
manipulated variables
input variables
process
sensors
actuators
controlled variables

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?

41



Examples of Process Control

Return air
Furnace
Hot air
Thermonstat
Temperature sensor
Gas-valve control
Temperature-setting control
user timing constraints
Timing constraints
controller
Circuit
Speed?
Power?

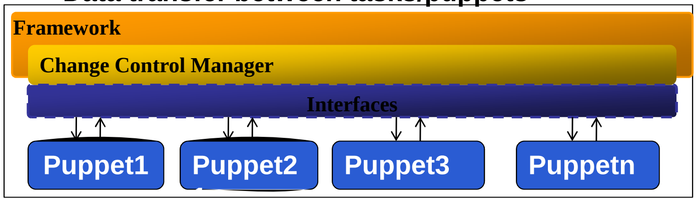
Process control structural pattern

Launching transformations

42

Pattern 9: Puppeteer

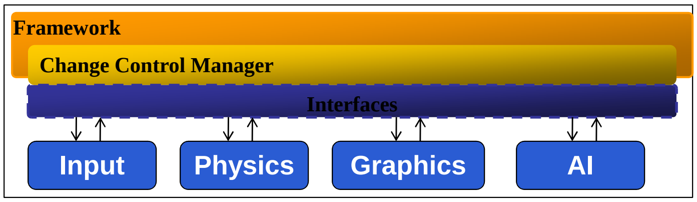
- Need an efficient way to manage and control the interaction of multiple simulators/computational agents
- Puppeteer Pattern – guides the interaction between the tasks/puppets to guarantee correctness of the overall task
- Puppeteer: 1) schedules puppets 2) manages exchange of data between puppets
- Difference with agent and repository?
  - No central repository
  - Data transfer between tasks/puppets



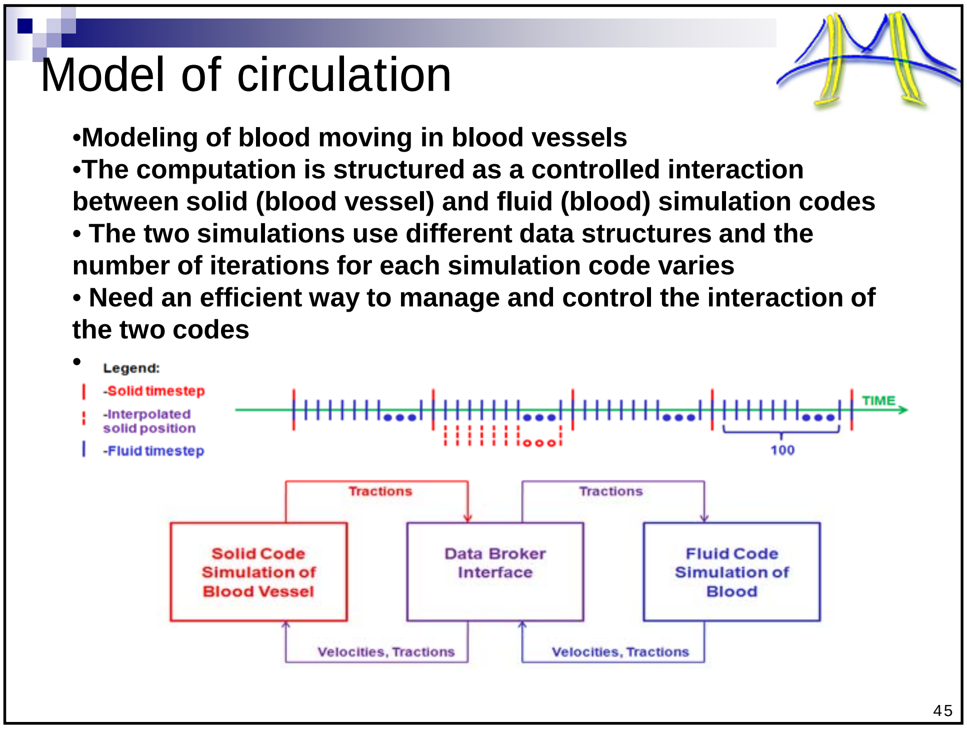
Examples?

43/17

Video Game



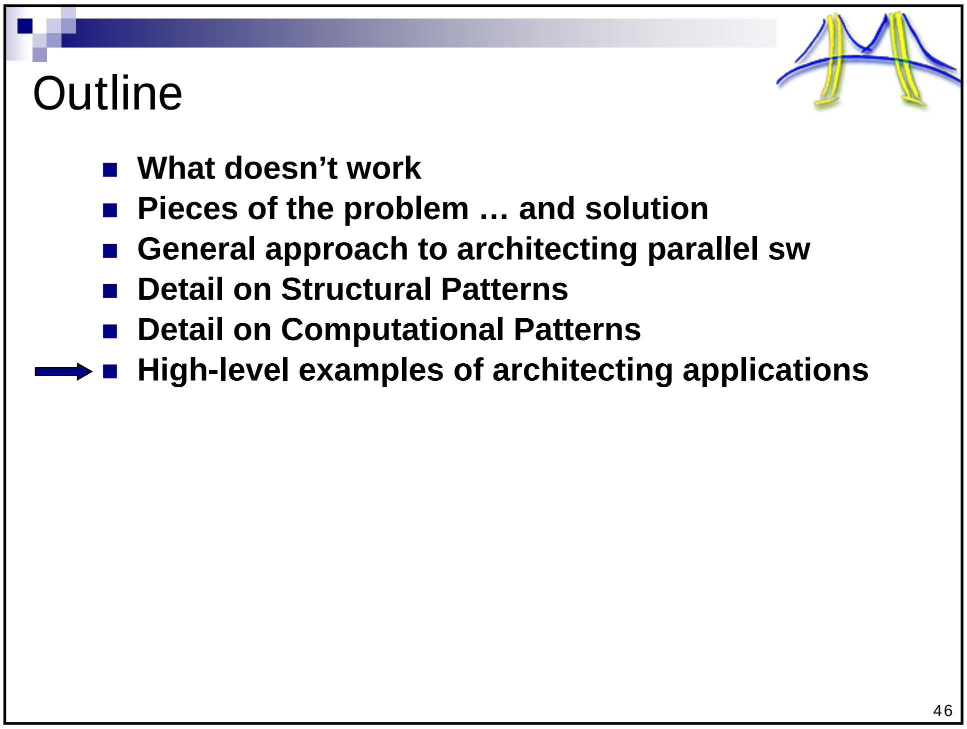
44/17



Model of circulation

- •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
- •

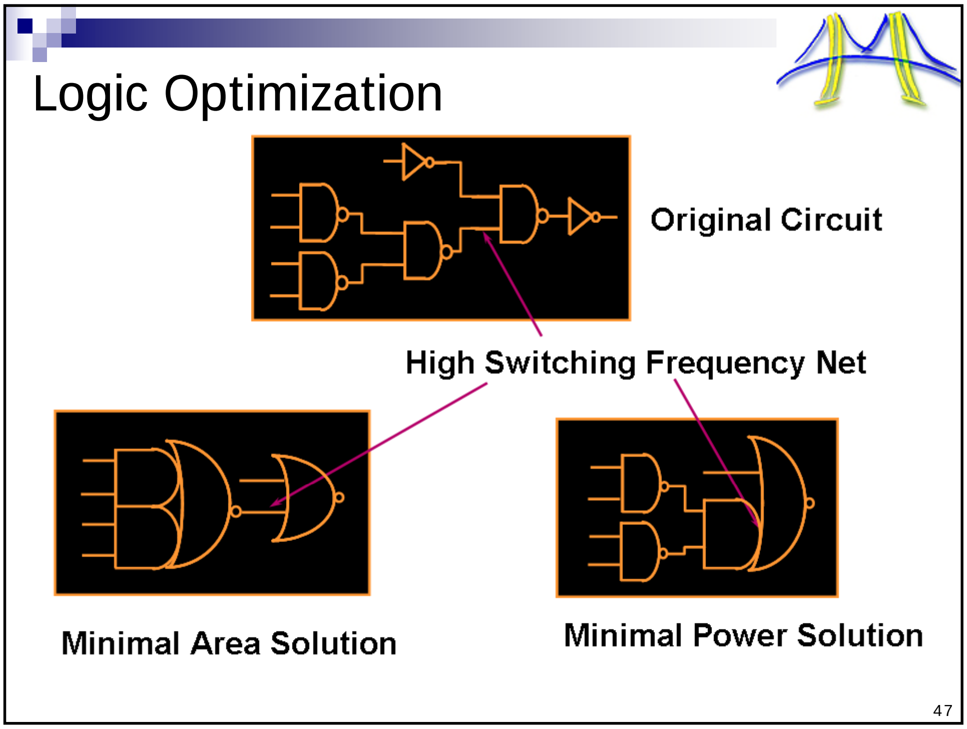
45



Outline

- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

46



Logic Optimization

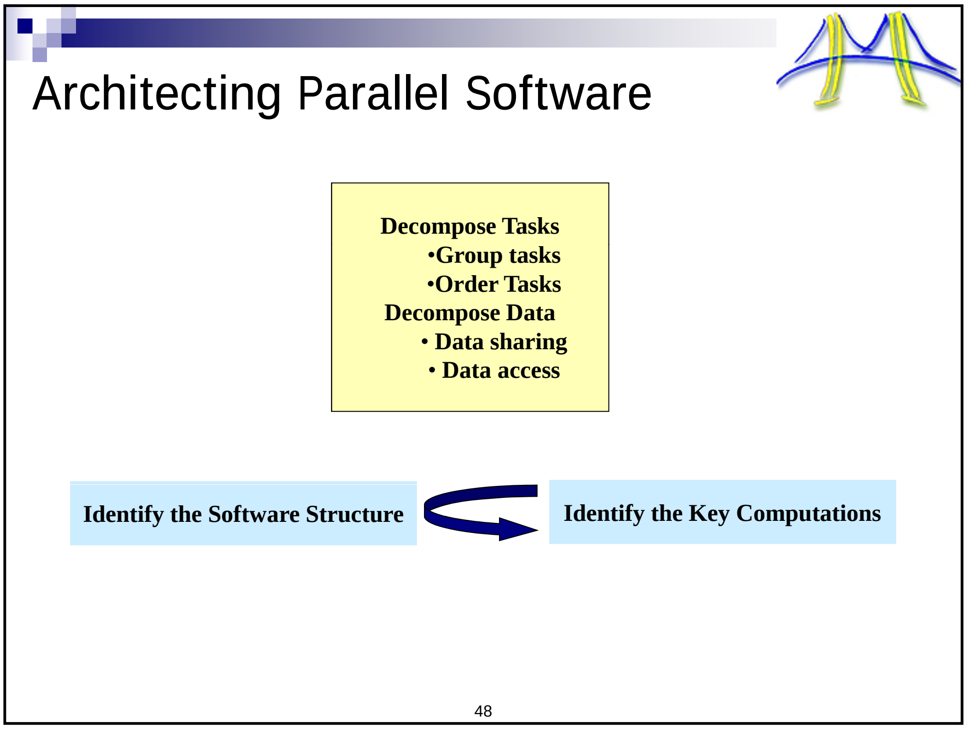
Original Circuit

High Switching Frequency Net

Minimal Area Solution

Minimal Power Solution

47



Architecting Parallel Software

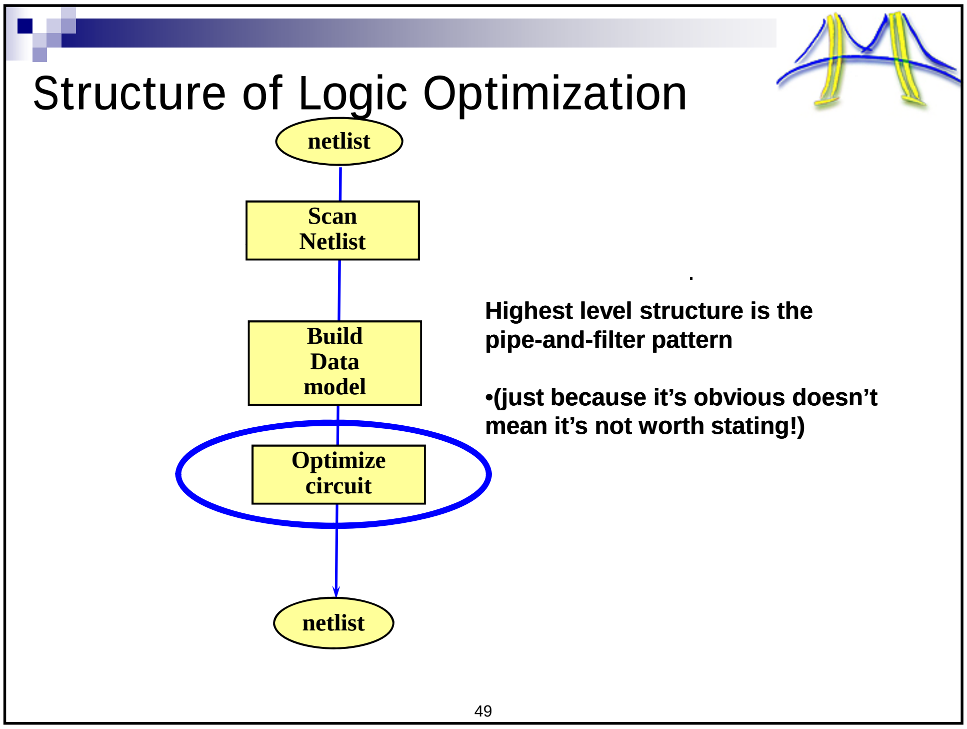
Decompose Tasks
- **Group tasks**
- **Order Tasks**

Decompose Data
- **Data sharing**
- **Data access**

Identify the Software Structure

Identify the Key Computations

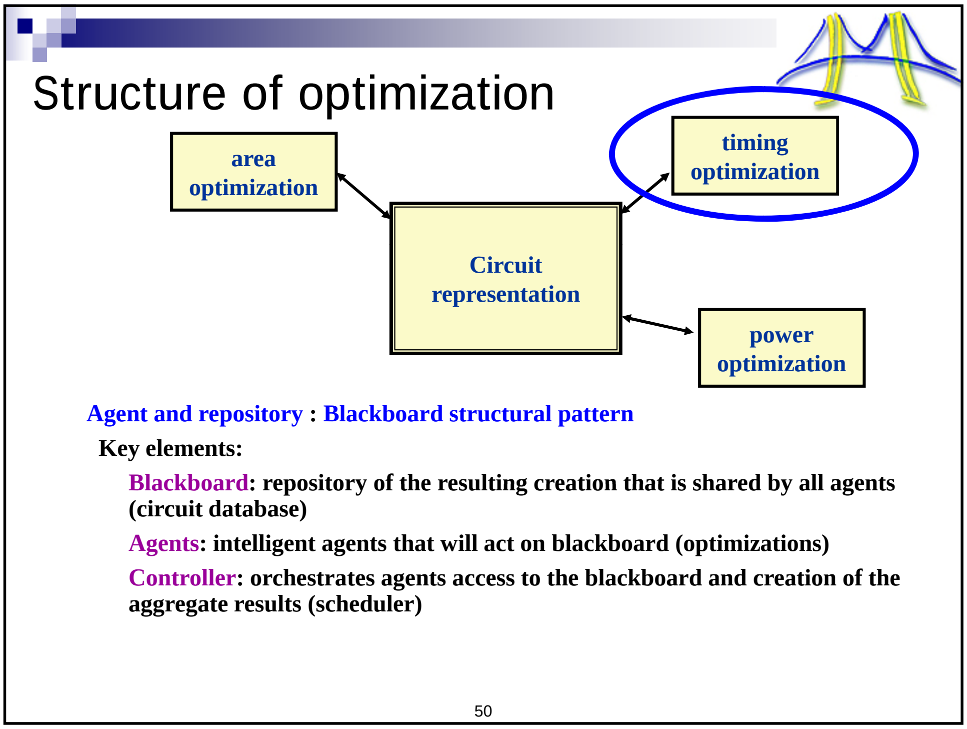
48



Structure of Logic Optimization

Highest level structure is the pipe-and-filter pattern

- (just because it’s obvious doesn’t mean it’s not worth stating!)



Structure of optimization

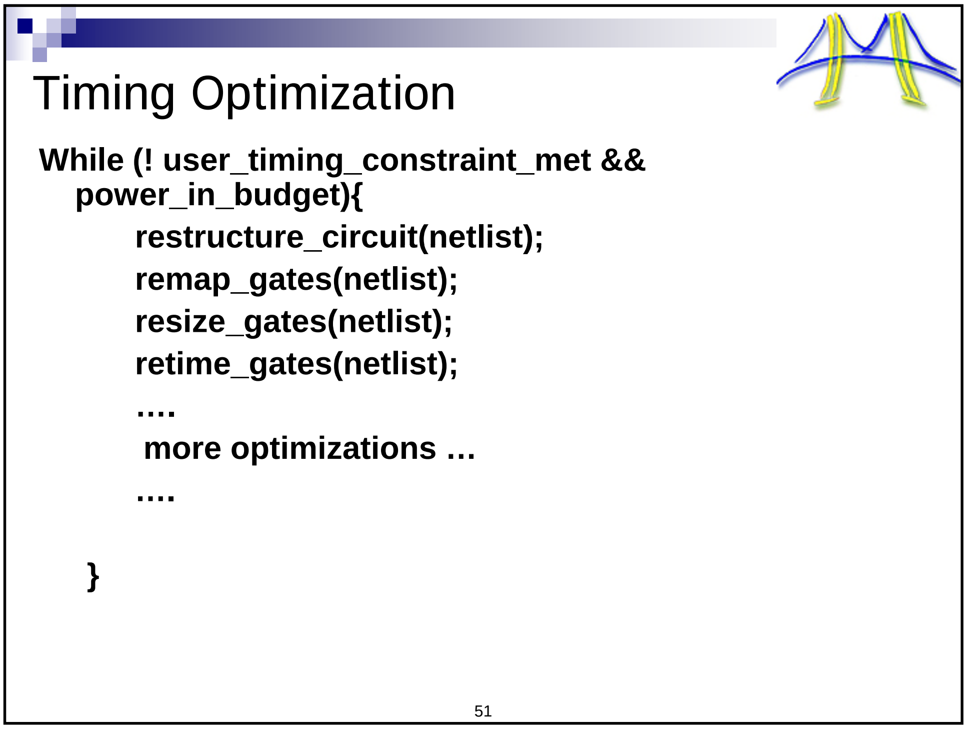
Agent and repository : Blackboard structural pattern

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)

Controller: orchestrates agents access to the blackboard and creation of the aggregate results (scheduler)

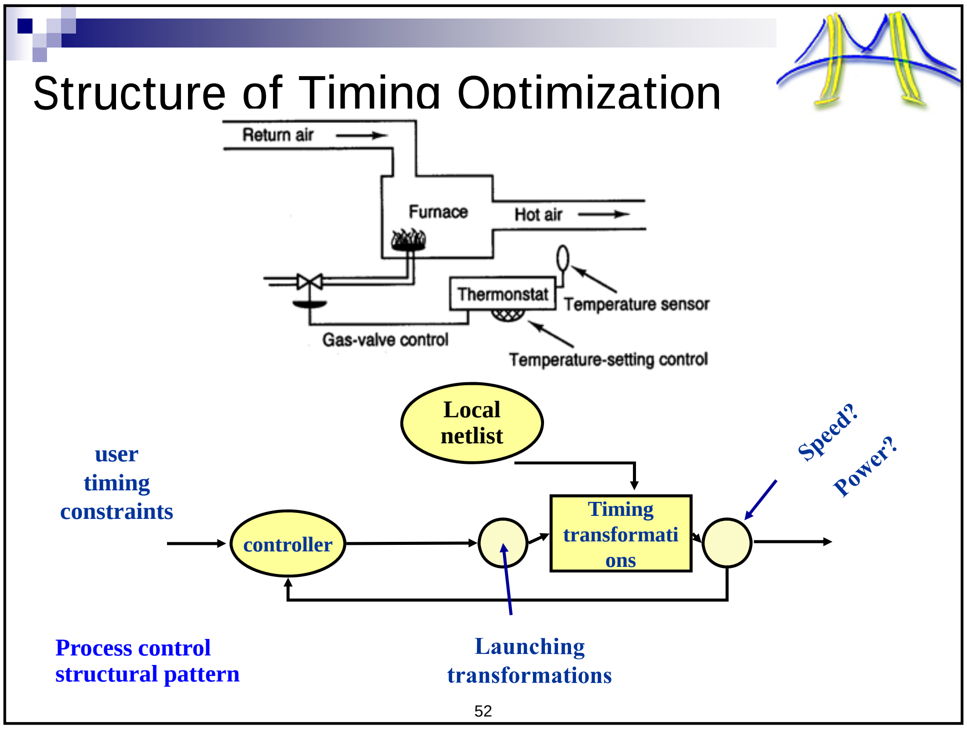


Timing Optimization

```
While (! user_timing_constraint_met &&
   power_in_budget){
        restructure_circuit(netlist);
        remap_gates(netlist);
        resize_gates(netlist);
        retime_gates(netlist);
        ….
         more optimizations …
        ….

   }
```

51



Structure of Timing Optimization

Local netlist

user timing constraints

controller

Timing transformations

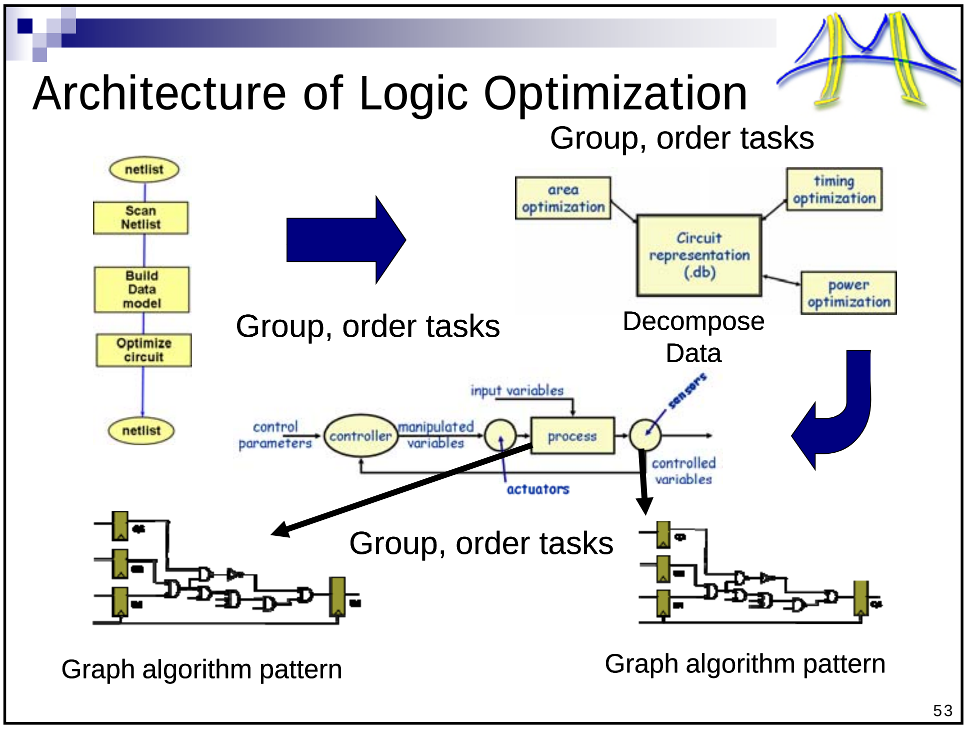
Speed?

Power?

Process control structural pattern

Launching transformations

52



Architecture of Logic Optimization

Group, order tasks

Group, order tasks

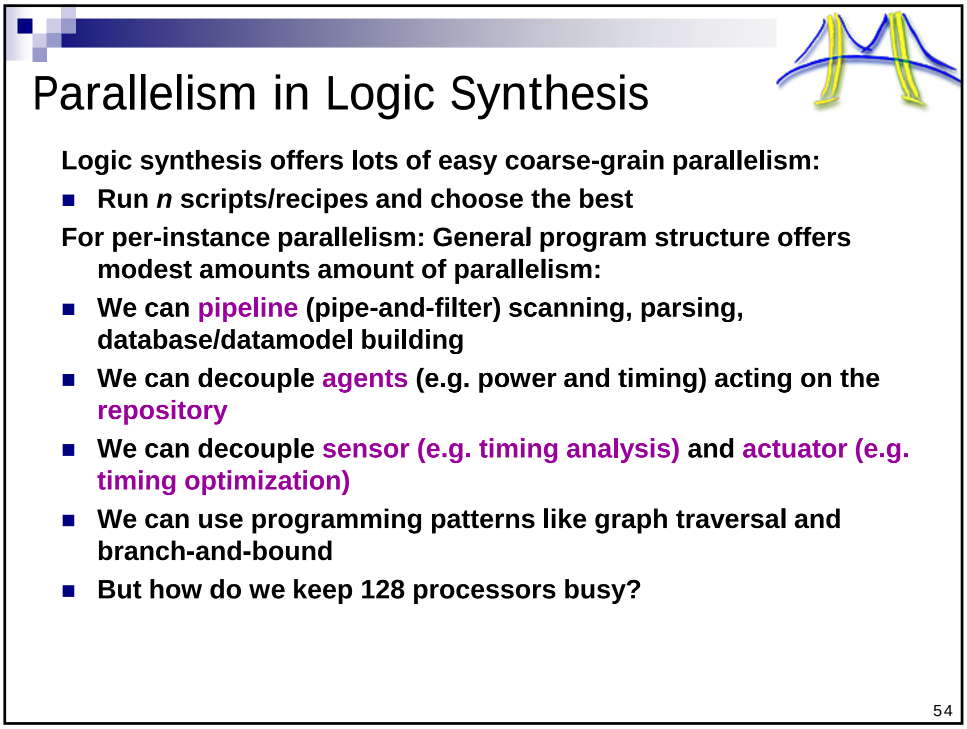
Decompose Data

Group, order tasks

Graph algorithm pattern

Graph algorithm pattern

53



Parallelism in Logic Synthesis

Logic synthesis offers lots of easy coarse-grain parallelism:

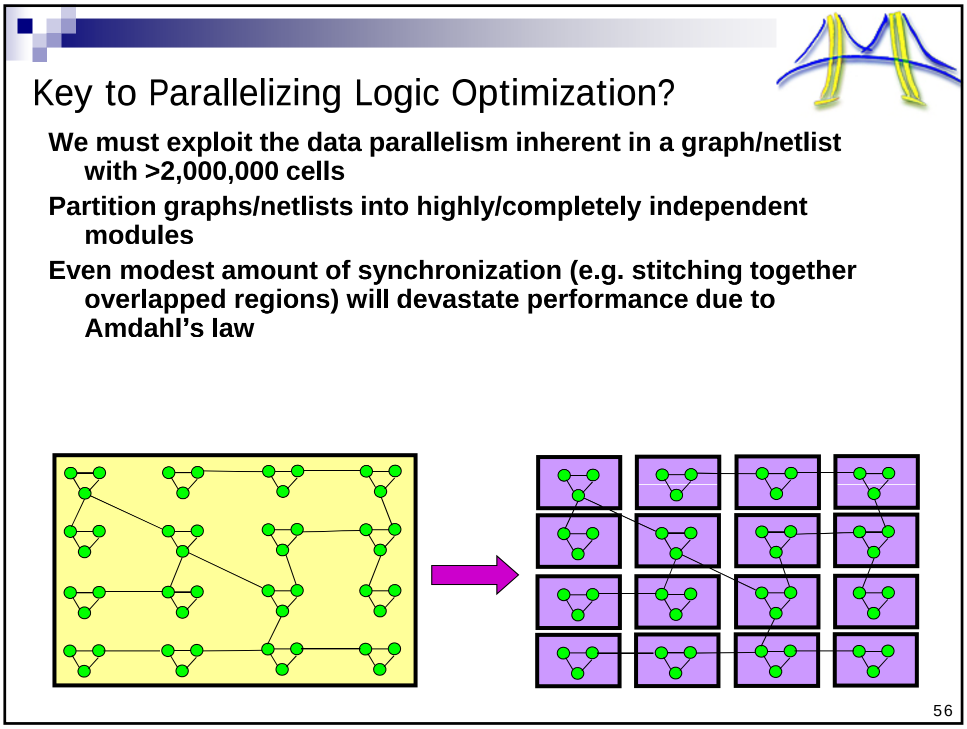
- **Run *n* scripts/recipes and choose the best**

For per-instance parallelism: General program structure offers modest amounts amount of parallelism:

- **We can pipeline (pipe-and-filter) scanning, parsing, database/datamodel building**
- **We can decouple agents (e.g. power and timing) acting on the repository**
- **We can decouple sensor (e.g. timing analysis) and actuator (e.g. timing optimization)**
- **We can use programming patterns like graph traversal and branch-and-bound**
- **But how do we keep 128 processors busy?**

54

Here’s a hint …

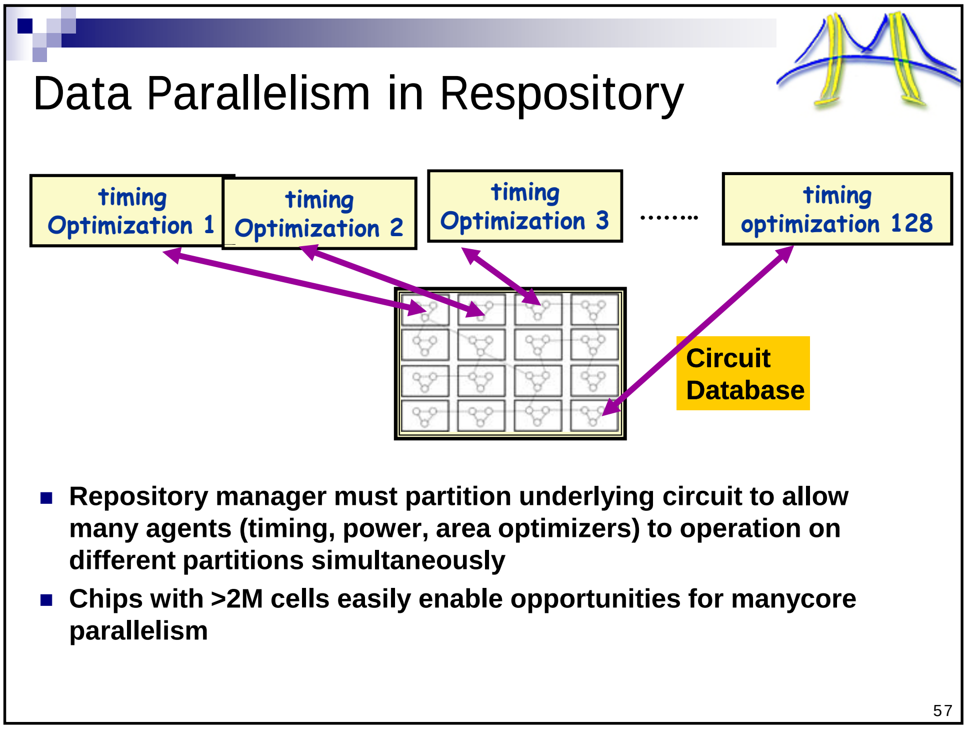


Key to Parallelizing Logic Optimization?

We must exploit the data parallelism inherent in a graph/netlist with >2,000,000 cells

Partition graphs/netlists into highly/completely independent modules

Even modest amount of synchronization (e.g. stitching together overlapped regions) will devastate performance due to Amdahl’s law



Data Parallelism in Respository

timing Optimization 1

timing Optimization 2

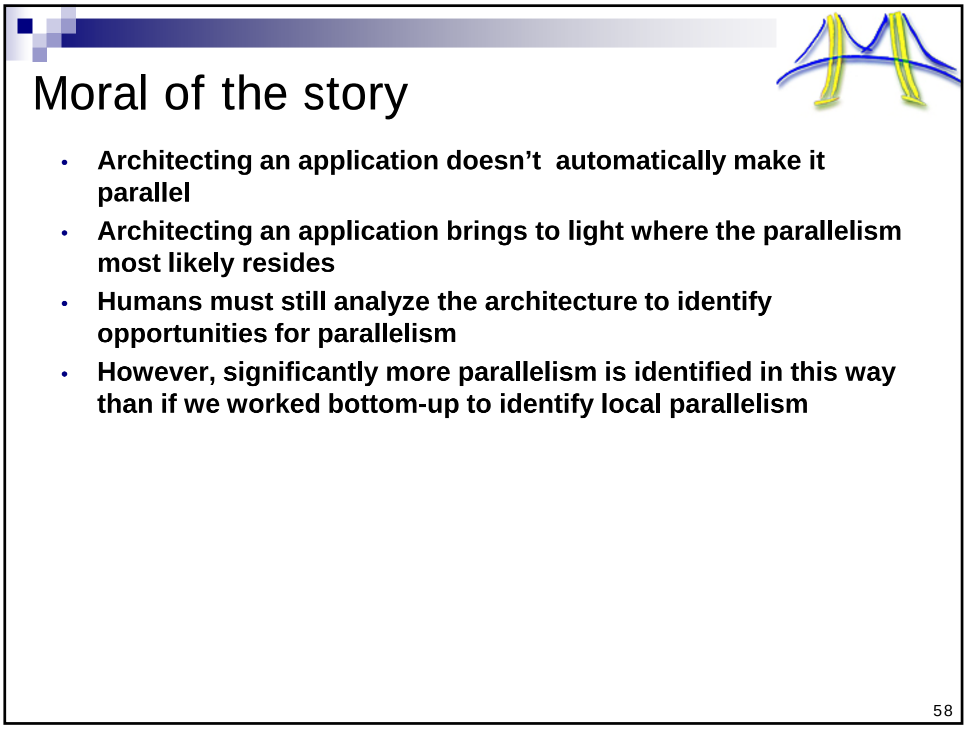
timing Optimization 3

……..

timing optimization 128

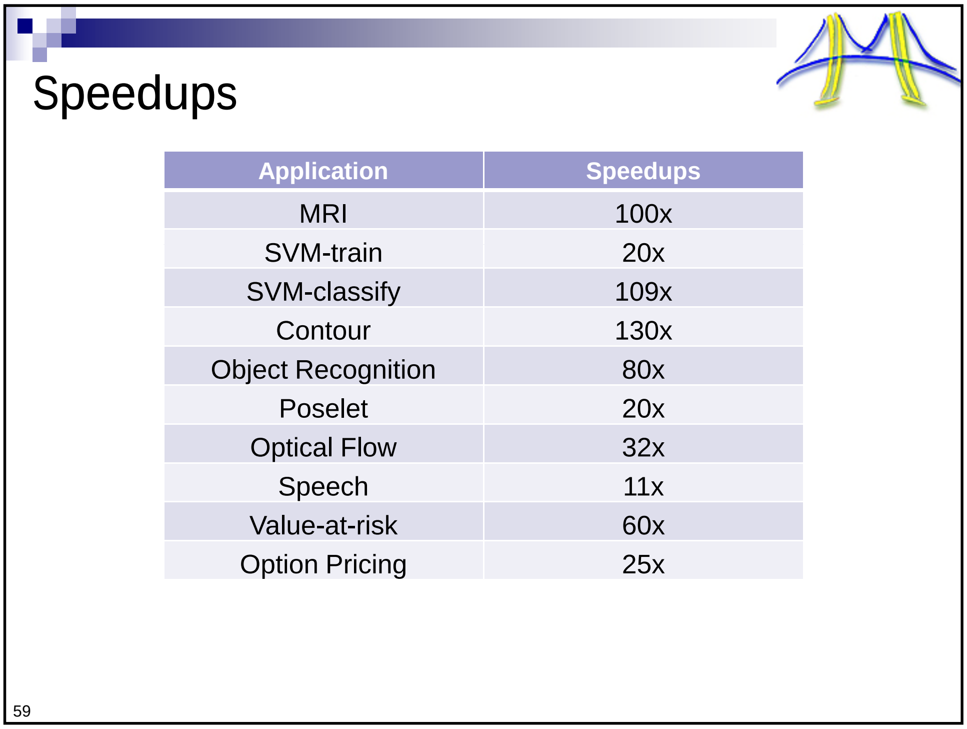
Circuit Database

- Repository manager must partition underlying circuit to allow many agents (timing, power, area optimizers) to operation on different partitions simultaneously
- Chips with >2M cells easily enable opportunities for manycore parallelism



Moral of the story

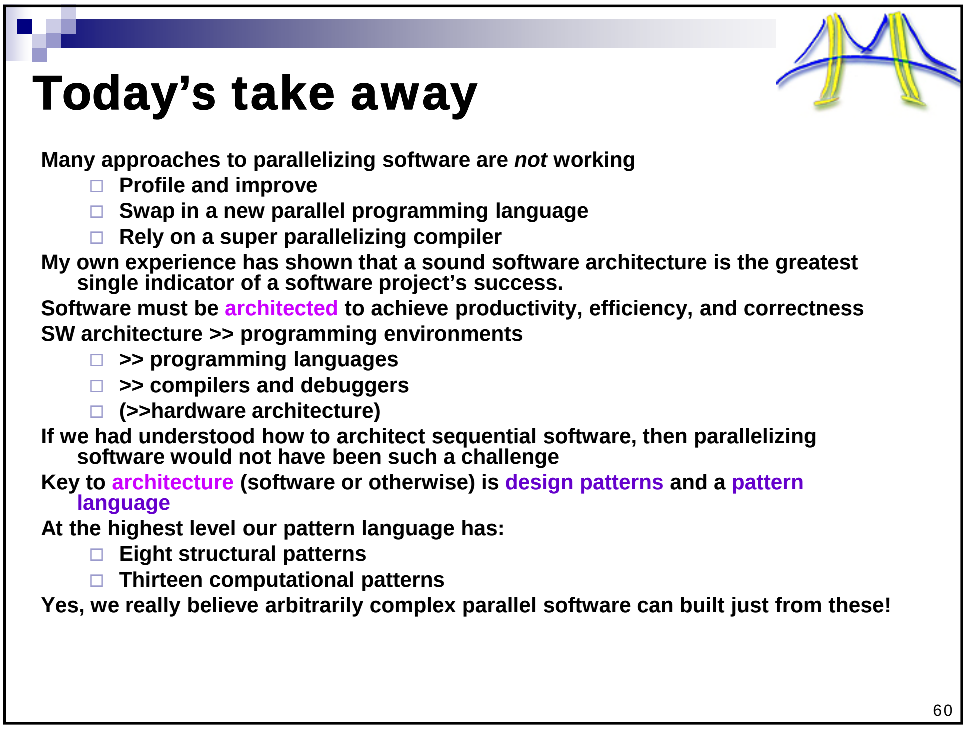
- Architecting an application doesn’t automatically make it parallel
- Architecting an application brings to light where the parallelism most likely resides
- Humans must still analyze the architecture to identify opportunities for parallelism
- However, significantly more parallelism is identified in this way than if we worked bottom-up to identify local parallelism



Speedups

Application	Speedups
MRI	100x
SVM-train	20x
SVM-classify	109x
Contour	130x
Object Recognition	80x
Poselet	20x
Optical Flow	32x
Speech	11x
Value-at-risk	60x
Option Pricing	25x

59



Today’s take away

Many approaches to parallelizing software are *not* working

- □ **Profile and improve**
- □ **Swap in a new parallel programming language**
- □ **Rely on a super parallelizing compiler**

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)**

If we had understood how to architect sequential software, then parallelizing software would not have been such a challenge

Key to architecture (software or otherwise) is design patterns and a pattern language

At the highest level our pattern language has:

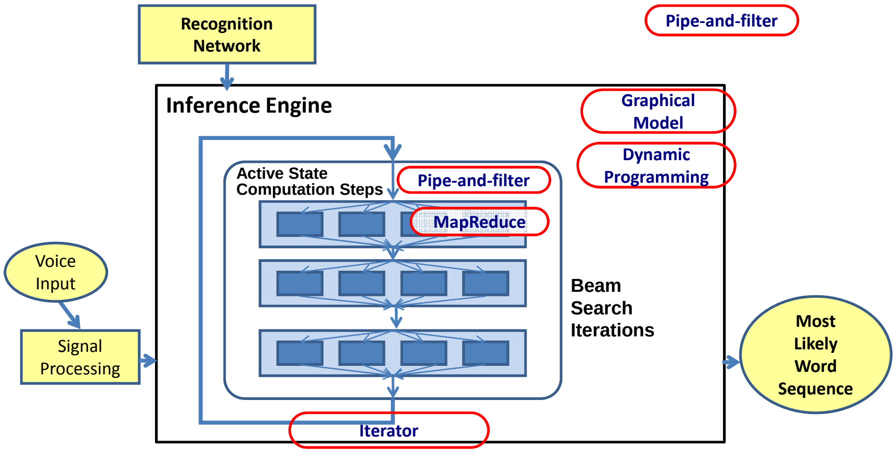
- □ **Eight structural patterns**
- □ **Thirteen computational patterns**

Yes, we really believe arbitrarily complex parallel software can built just from these!

60

More examples

Architecting Speech Recognition



Pipe-and-filter

Recognition Network

Inference Engine

Graphical Model

Dynamic Programming

Active State Computation Steps

Pipe-and-filter

MapReduce

Beam Search Iterations

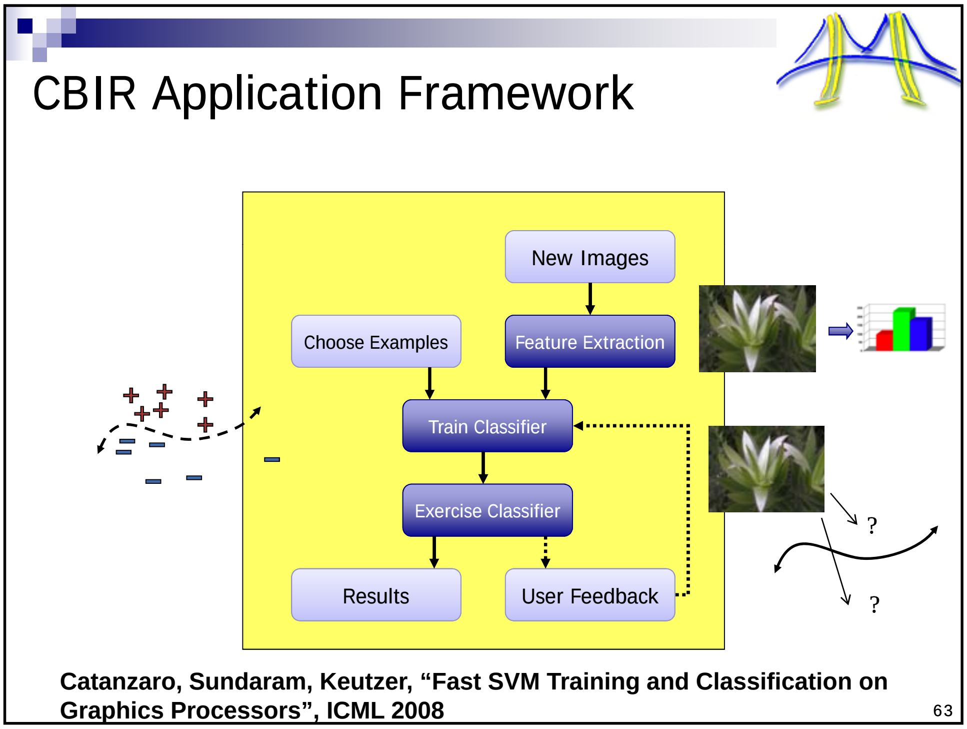
Voice Input

Signal Processing

Most Likely Word Sequence

Iterator

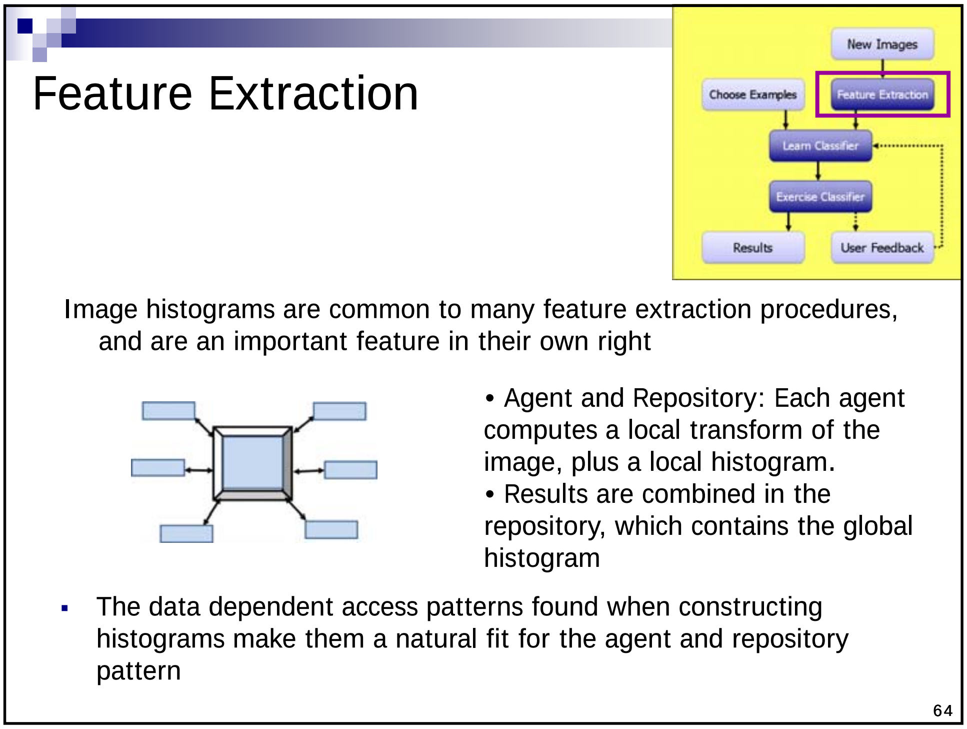
Large Vocabulary Continuous Speech Recognition Poster: Chong, Yi
Work also to appear at Emerging Applications for Manycore Architecture



CBIR Application Framework

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

63



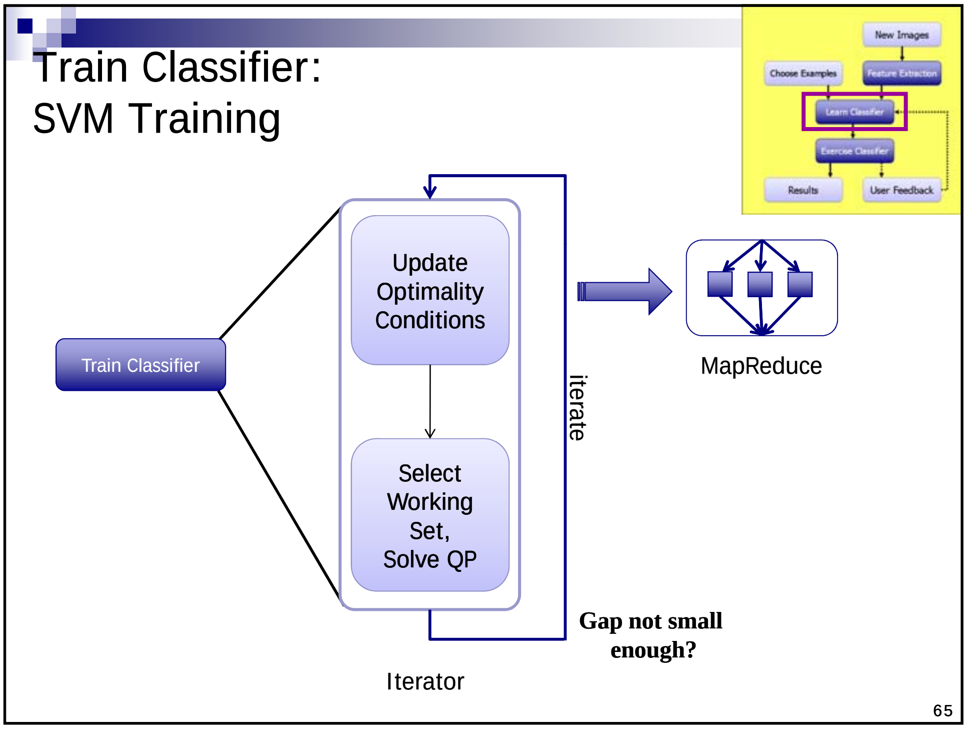
Feature Extraction

Image histograms are common to many feature extraction procedures, and are an important feature in their own right

- Agent and Repository: Each agent computes a local transform of the image, plus a local histogram.
- Results are combined in the repository, which contains the global histogram

- The data dependent access patterns found when constructing histograms make them a natural fit for the agent and repository pattern

64



Train Classifier: SVM Training

Train Classifier

Update Optimality Conditions

Select Working Set, Solve QP

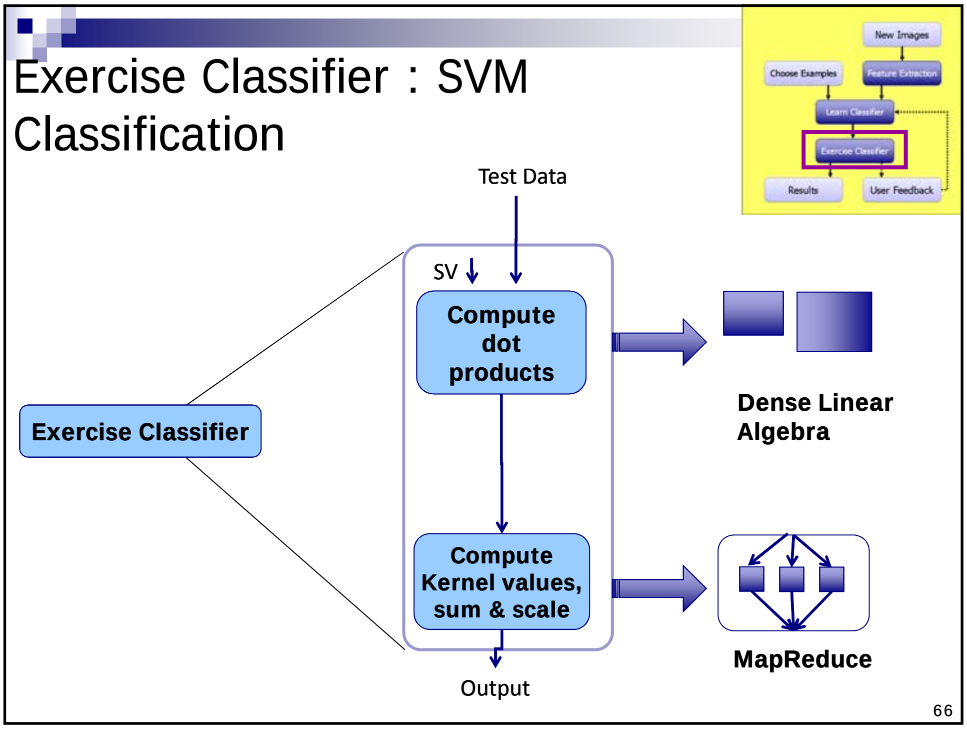
iterate

Gap not small enough?

Iterator

MapReduce

65



Exercise Classifier : SVM Classification

Test Data

SV

Compute dot products

Exercise Classifier

Compute Kernel values, sum & scale

Output

Dense Linear Algebra

MapReduce

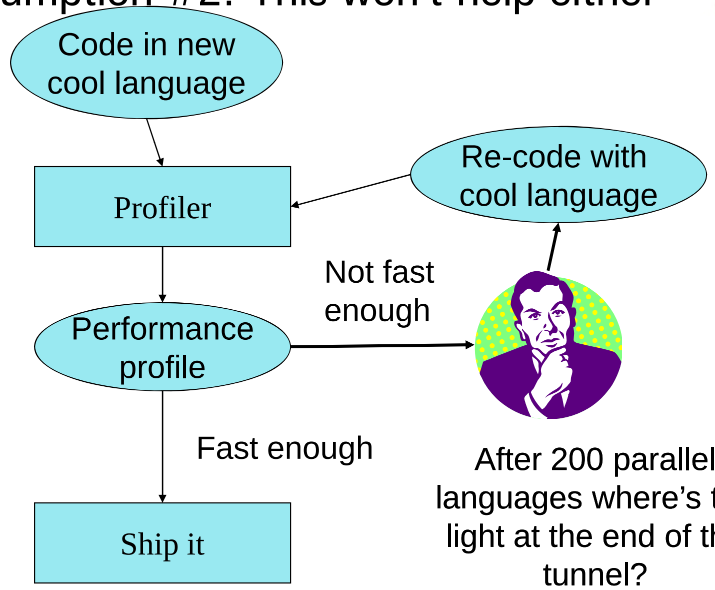
66

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

67

Assumption #2: This won’t help either


Code in new cool language

Profiler

Re-code with cool language

Not fast enough

Performance profile

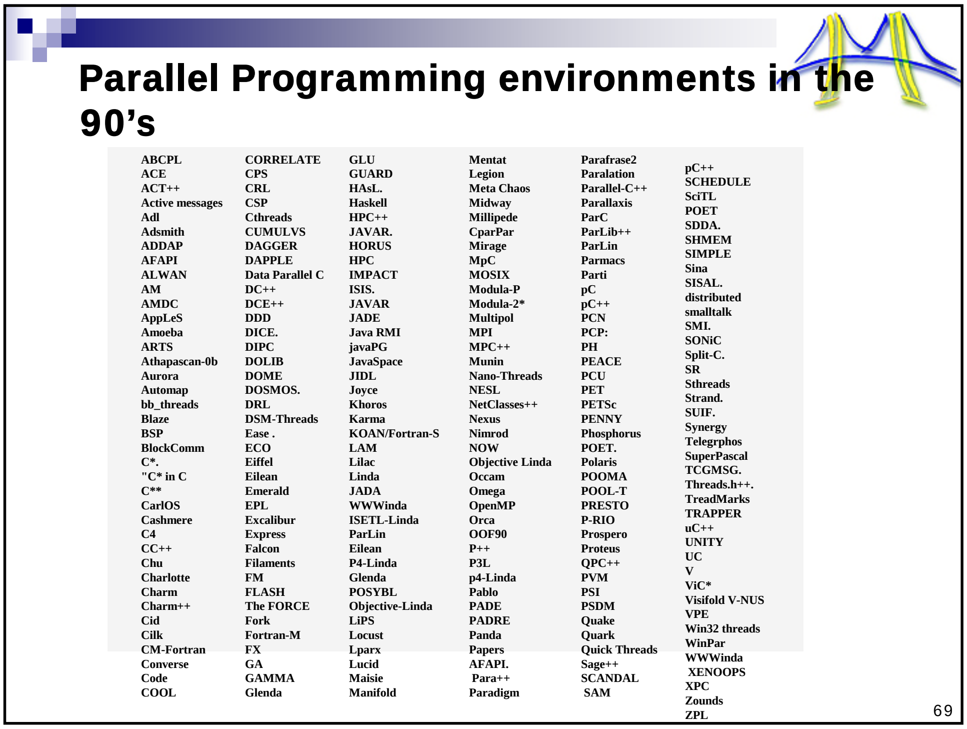
Fast enough

Ship it

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

68

68



Parallel Programming environments in the 90's

ABCPL
ACE
ACT++
Active messages
Adl
Adsmith
ADDAP
AFAPI
ALWAN
AM
AMDC
AppLeS
Amoeba
ARTS
Athapascan-0b
Aurora
Automap
bb_threads
Blaze
BSP
BlockComm
C*.
"C* in C
C**
CarlOS
Cashmere
C4
CC++
Chu
Charlotte
Charm
Charm++
Cid
Cilk
CM-Fortran
Converse
Code
COOL

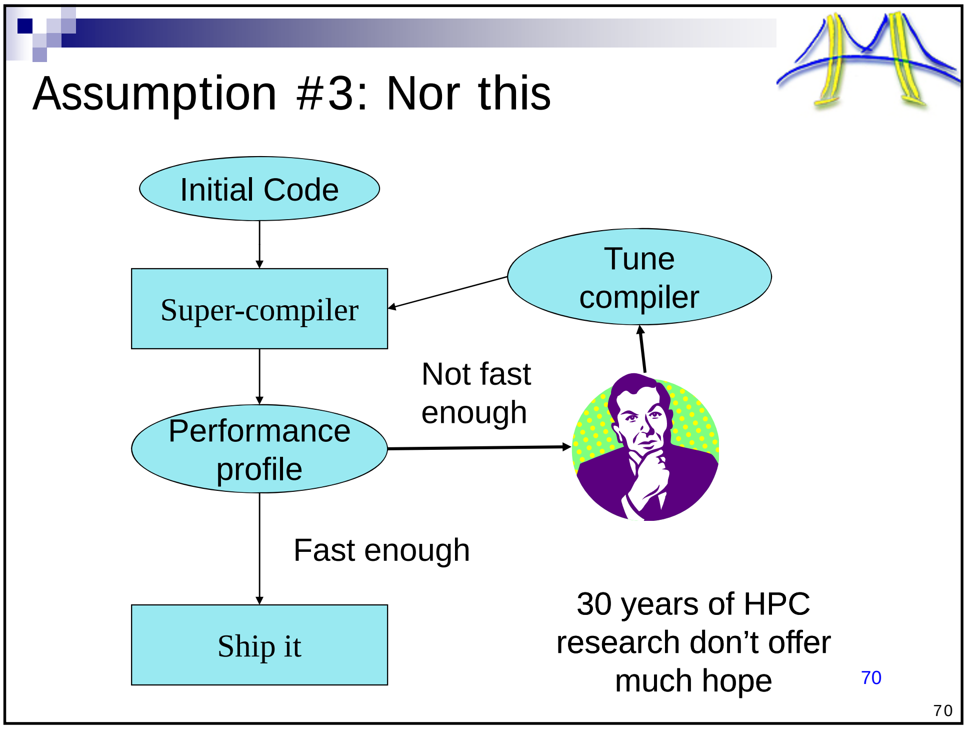
CORRELATE
CPS
CRL
CSP
Cthreads
CUMULVS
DAGGER
DAPPLE
Data Parallel C
DC++
DCE++
DDD
DICE.
DIPC
DOLIB
DOME
DOSMOS.
DRL
DSM-Threads
Ease .
ECO
Eiffel
Eilean
Emerald
EPL
Excalibur
Express
Falcon
Filaments
FM
FLASH
The FORCE
Fork
Fortran-M
FX
GA
GAMMA
Glenda

GLU
GUARD
HAsL.
Haskell
HPC++
JAVAR.
HORUS
HPC
IMPACT
ISIS.
JAVAR
JADE
Java RMI
javaPG
JavaSpace
JIDL
Joyce
Khoros
Karma
KOAN/Fortran-S
LAM
Lilac
Linda
JADA
WWWinda
ISETL-Linda
ParLin
Eilean
P4-Linda
Glenda
POSYBL
Objective-Linda
LiPS
Locust
Lparx
Lucid
Maisie
Manifold

Mentat
Legion
Meta Chaos
Midway
Millipede
CparPar
Mirage
MpC
MOSIX
Modula-P
Modula-2*
Multipol
MPI
MPC++
Munin
Nano-Threads
NESL
NetClasses++
Nexus
Nimrod
NOW
Objective Linda
Occam
Omega
OpenMP
Orca
OOF90
P++
P3L
p4-Linda
Pablo
PADE
PADRE
Panda
Papers
AFAPI.
Para++
Paradigm

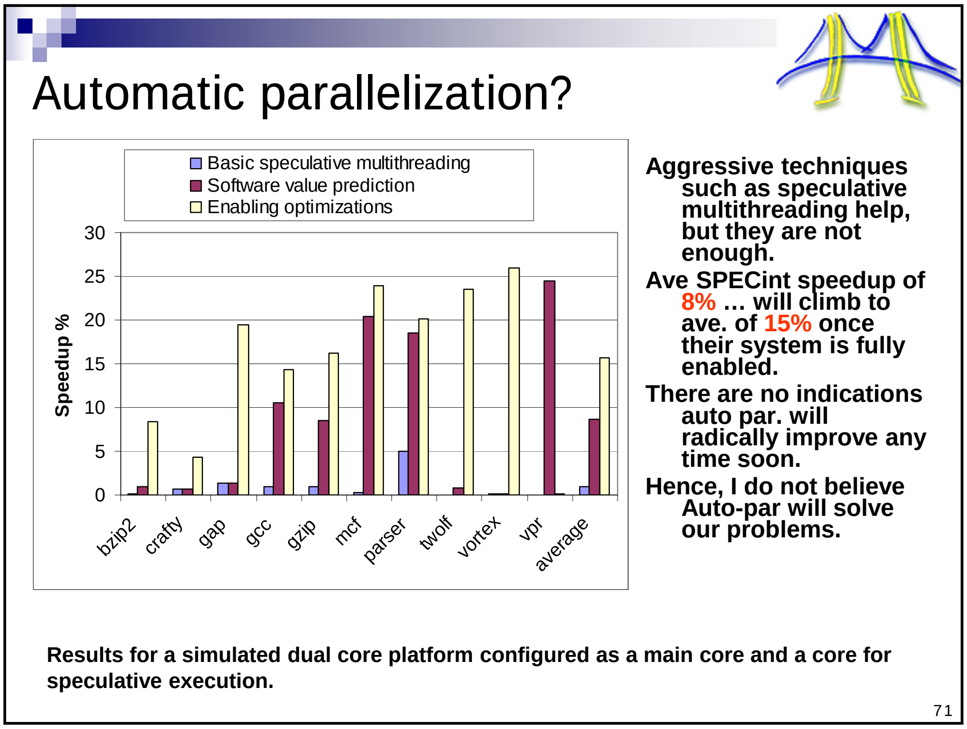
Parafrase2
Paralation
Parallel-C++
Parallaxis
ParC
ParLib++
ParLin
Parmacs
Parti
pC
pC++
PCN
PCP:
PH
PEACE
PCU
PET
PETSc
PENNY
Phosphorus
POET.
Polaris
POOMA
POOL-T
PRESTO
P-RIO
Prospero
Proteus
QPC++
PVM
PSI
PSDM
Quake
Quark
Quick Threads
Sage++
SCANDAL
SAM

pC++
SCHEDULE
SciTL
POET
SDDA.
SHMEM
SIMPLE
Sina
SISAL.
distributed
smalltalk
SMI.
SONiC
Split-C.
SR
Sthreads
Strand.
SUIF.
Synergy
Telegrphos
SuperPascal
TCGMSG.
Threads.h++.
TreadMarks
TRAPPER
uC++
UNITY
UC
V
ViC*
Visifold V-NUS
VPE
Win32 threads
WinPar
WWWinda
XENOOPS
XPC
Zounds
ZPL



Assumption #3: Nor this

30 years of HPC research don't offer much hope



Automatic parallelization?

Aggressive techniques such as speculative multithreading help, but they are not enough.

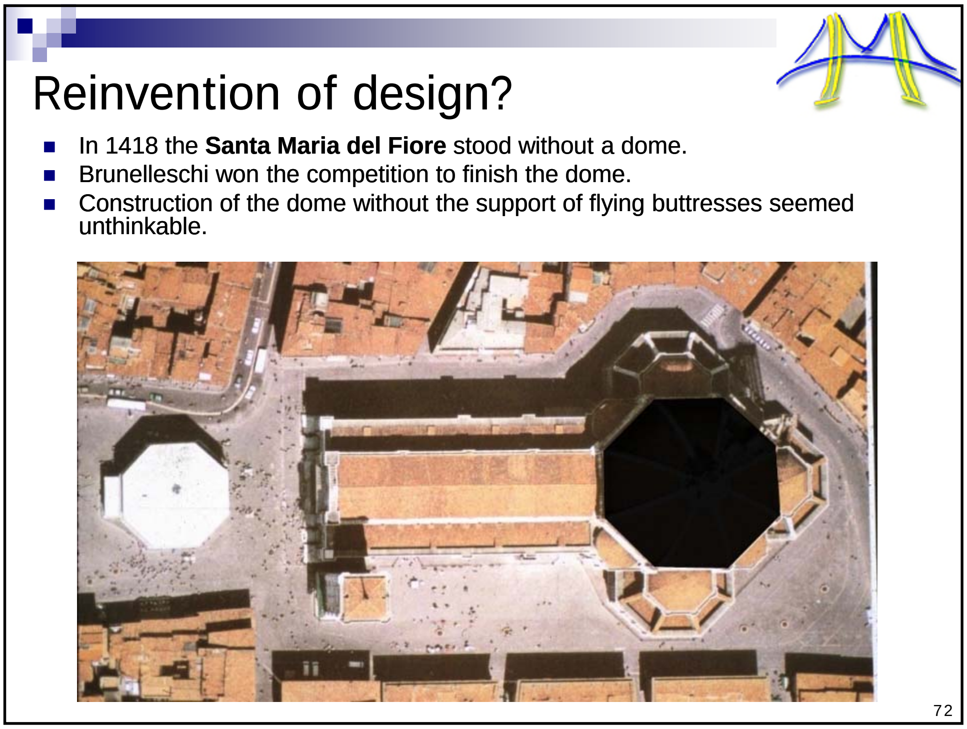
Ave SPECint speedup of 8% … will climb to ave. of 15% once their system is fully enabled.

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.

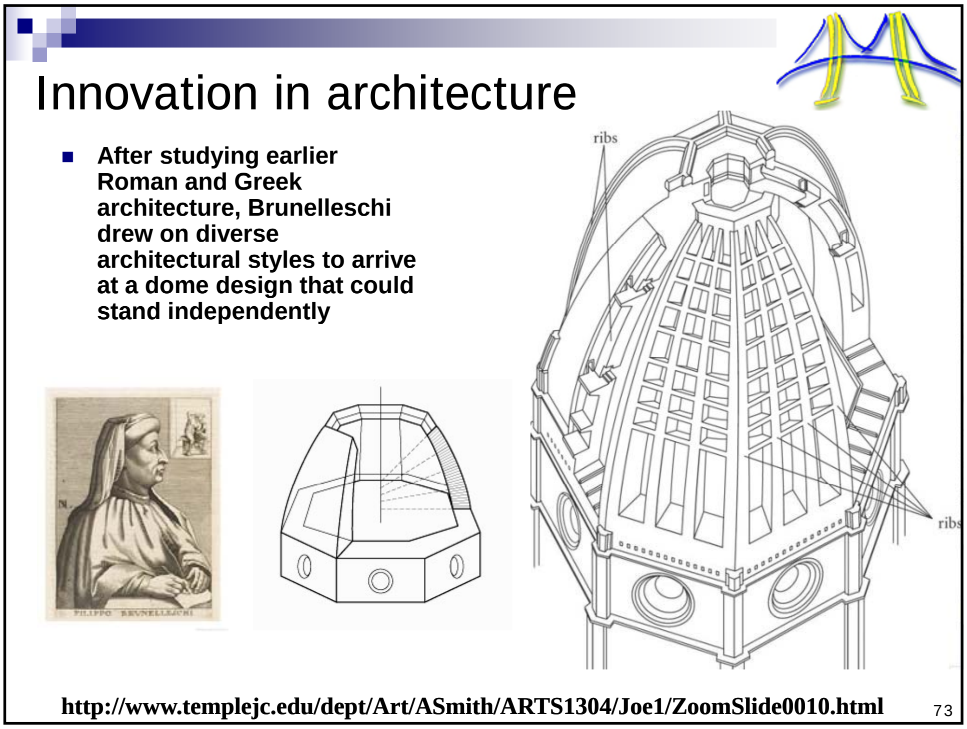
71



Reinvention of design?

- In 1418 the **Santa Maria del Fiore** stood without a dome.
- Brunelleschi won the competition to finish the dome.
- Construction of the dome without the support of flying buttresses seemed unthinkable.

72

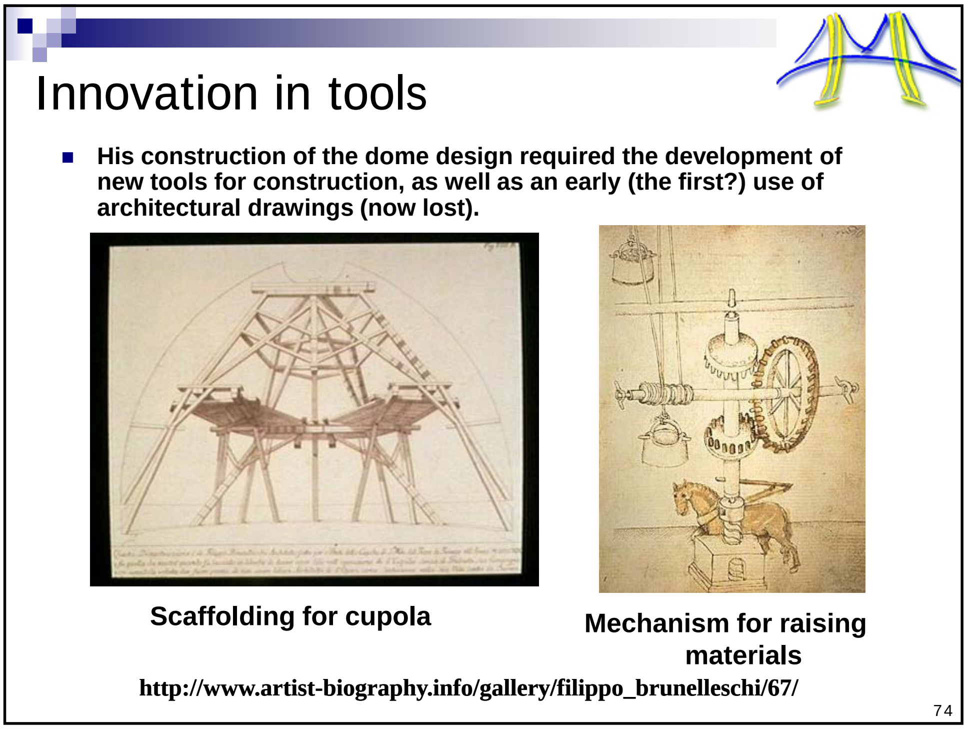


Innovation in architecture

- After studying earlier Roman and Greek architecture, Brunelleschi drew on diverse architectural styles to arrive at a dome design that could stand independently

ribs
ribs
N.
FILIPPO BRUNELLESCHI

http://www.templejc.edu/dept/Art/ASmith/ARTS1304/Joe1/ZoomSlide0010.html



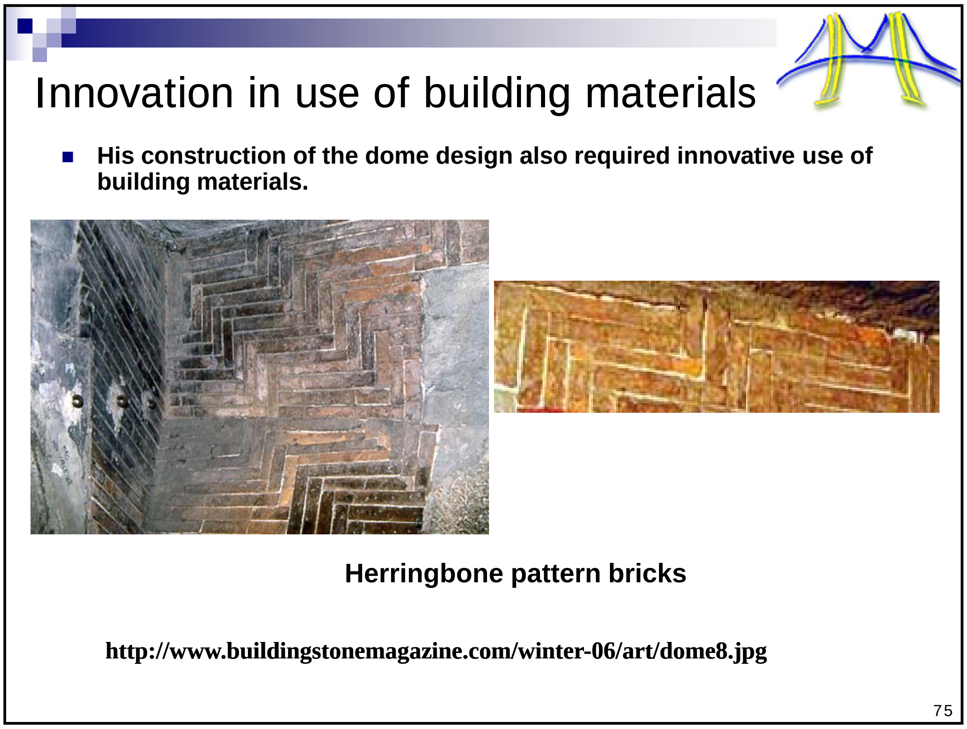
Innovation in tools

- His construction of the dome design required the development of new tools for construction, as well as an early (the first?) use of architectural drawings (now lost).

Scaffolding for cupola

Mechanism for raising materials

http://www.artist-biography.info/gallery/filippo_brunelleschi/67/



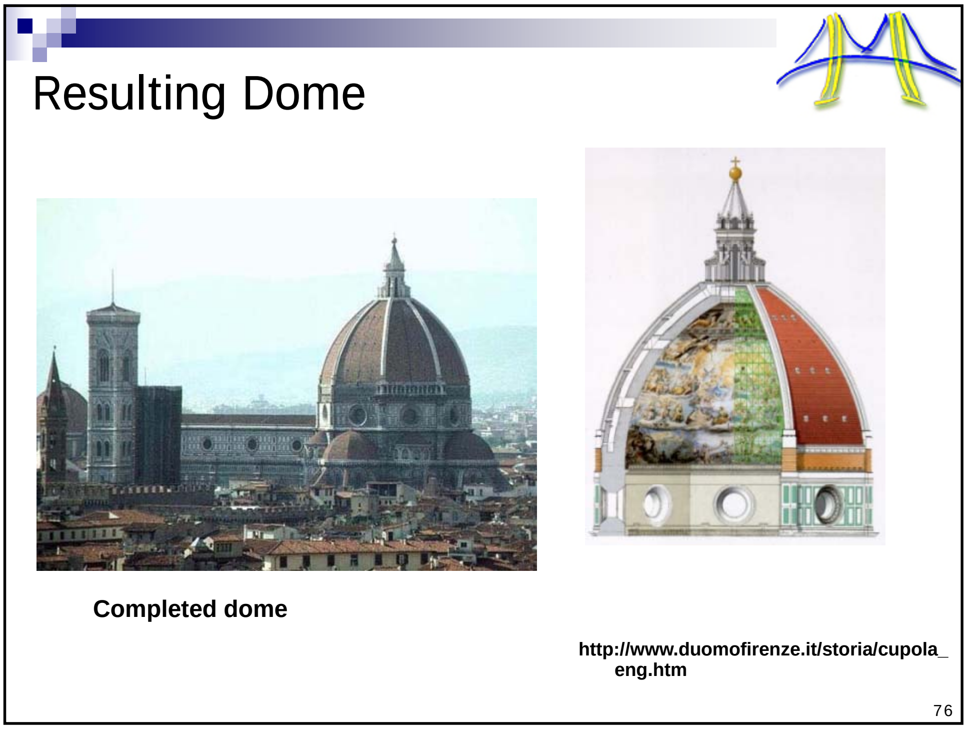
Innovation in use of building materials

- **His construction of the dome design also required innovative use of building materials.**

Herringbone pattern bricks

http://www.buildingstonemagazine.com/winter-06/art/dome8.jpg

75

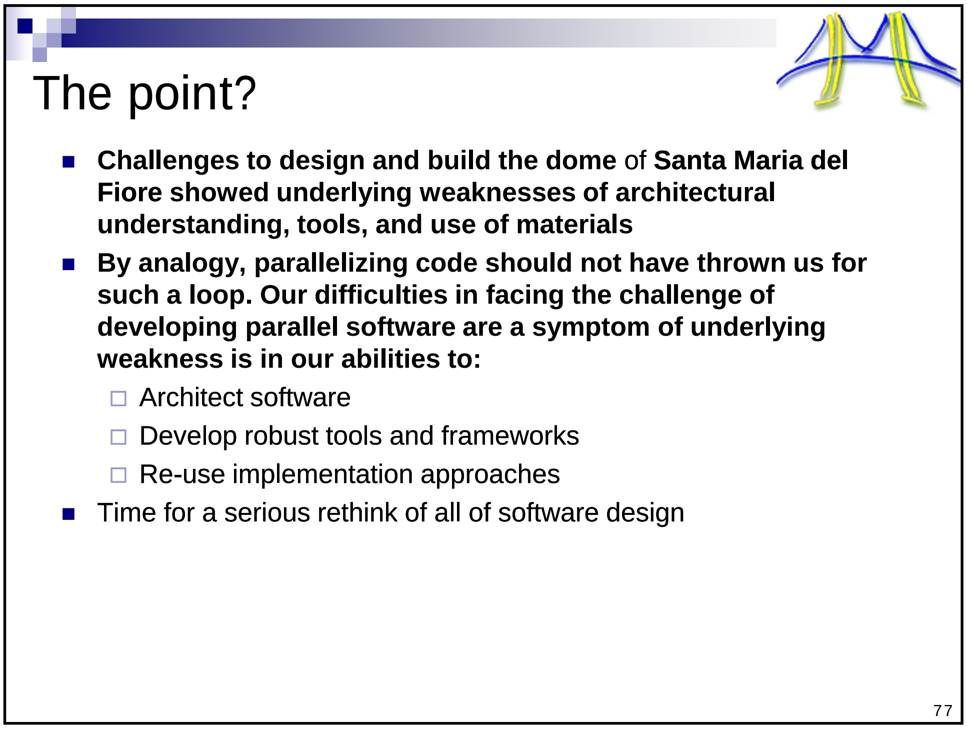


Resulting Dome

Completed dome

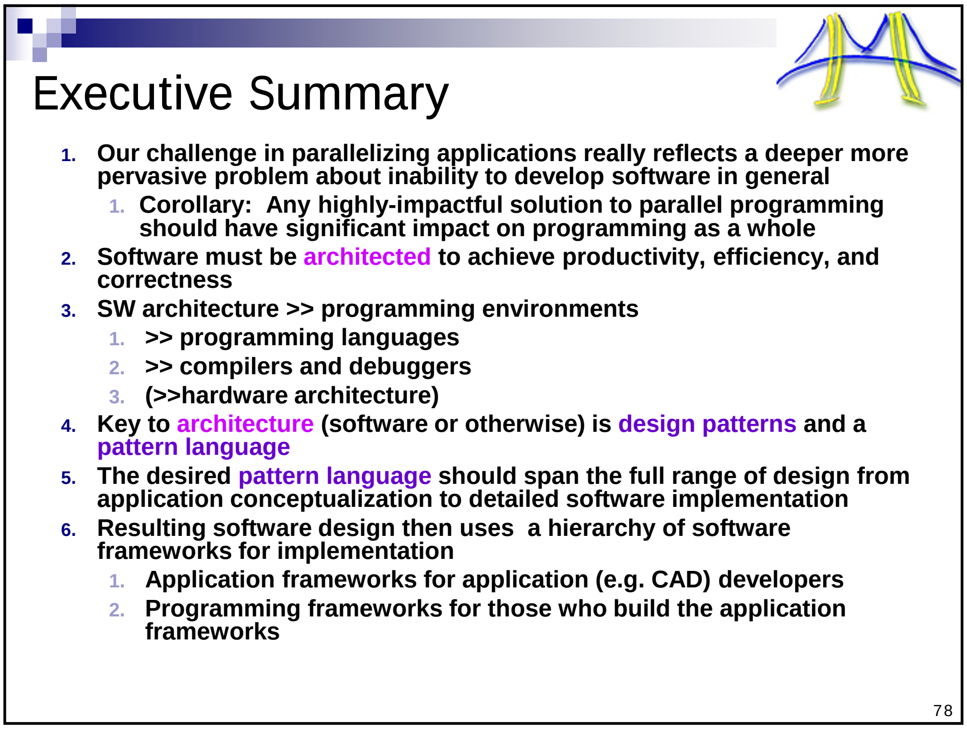
http://www.duomofirenze.it/storia/cupola_eng.htm

76



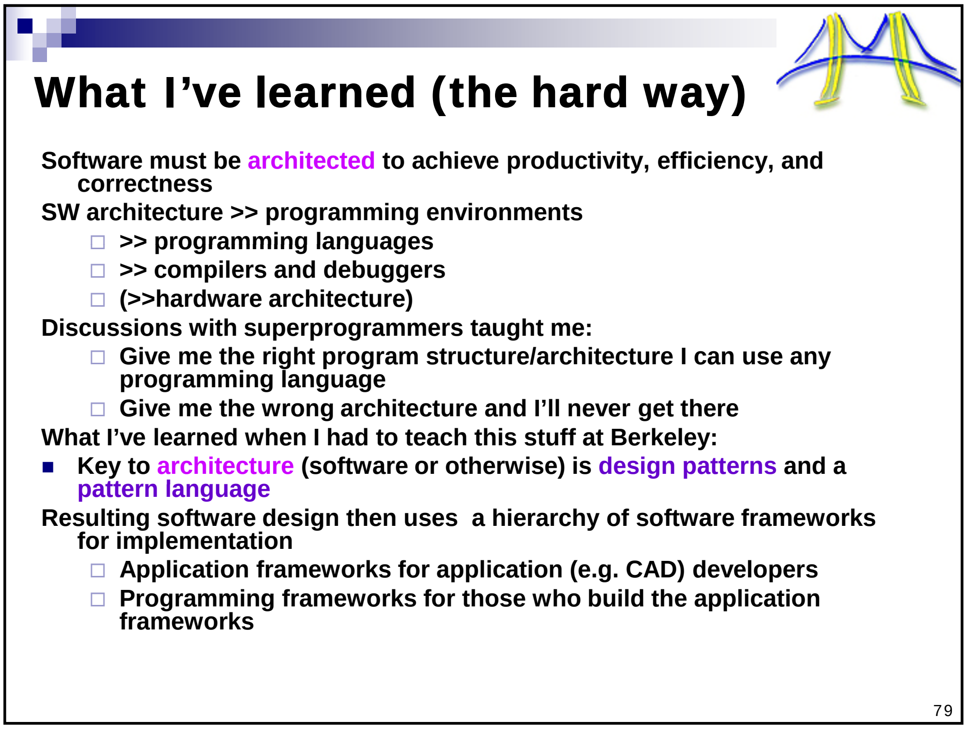
The point?

- **Challenges to design and build the dome** of **Santa Maria del Fiore showed underlying weaknesses of architectural understanding, tools, and use of materials**
- **By analogy, parallelizing code should not have thrown us for such a loop. Our difficulties in facing the challenge of developing parallel software are a symptom of underlying weakness is in our abilities to:**
  - □ Architect software
  - □ Develop robust tools and frameworks
  - □ Re-use implementation approaches
- Time for a serious rethink of all of software design



Executive Summary

1. **Our challenge in parallelizing applications really reflects a deeper more pervasive problem about inability to develop software in general**
   1. **Corollary: Any highly-impactful solution to parallel programming should have significant impact on programming as a whole**
2. **Software must be architected to achieve productivity, efficiency, and correctness**
3. **SW architecture >> programming environments**
   1. **>> programming languages**
   2. **>> compilers and debuggers**
   3. **(>>hardware architecture)**
4. **Key to architecture (software or otherwise) is design patterns and a pattern language**
5. **The desired pattern language should span the full range of design from application conceptualization to detailed software implementation**
6. **Resulting software design then uses a hierarchy of software frameworks for implementation**
   1. **Application frameworks for application (e.g. CAD) developers**
   2. **Programming frameworks for those who build the application frameworks**



What I’ve 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

79