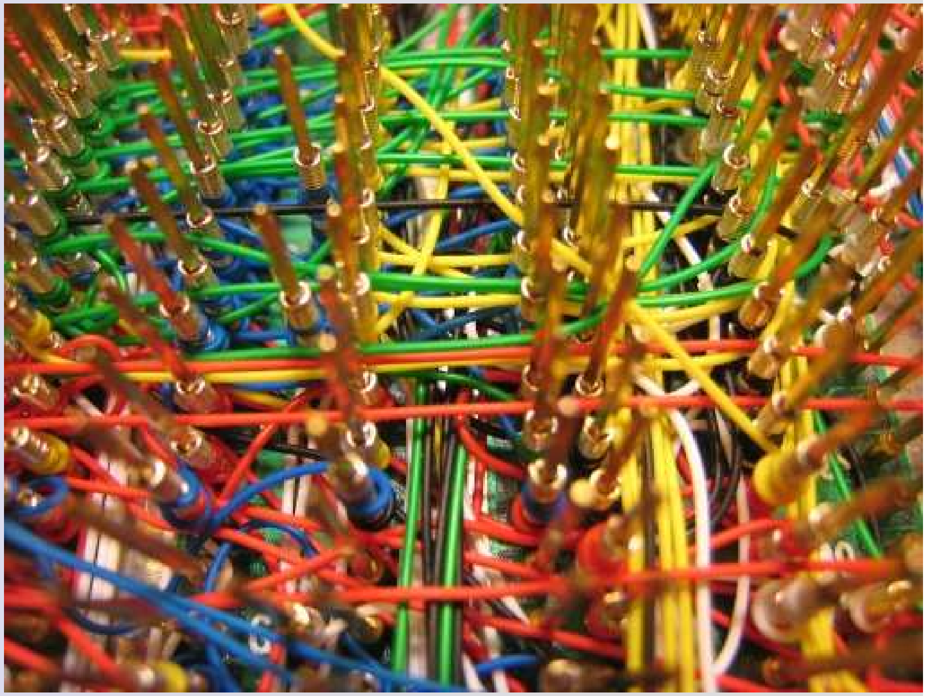
The Language, Optimizer, and Tools Mess

Erik Altman

April 4, 2011

Outline

- The Mess
- Optimizing the Mess
- Fixing the Mess



Caveat: This presentation contains my opinions.

No endorsement by IBM of the views expressed herein should be inferred.

Performance Mess: Slow Video Editing

YouTube Video Editor Brings Painfully Limited & Slow Video Editing To Everyone

Jun 16th, 2010 | By James Lewin

YouTube has added a new cloud-based Video Editor that brings basic video editing everyone.

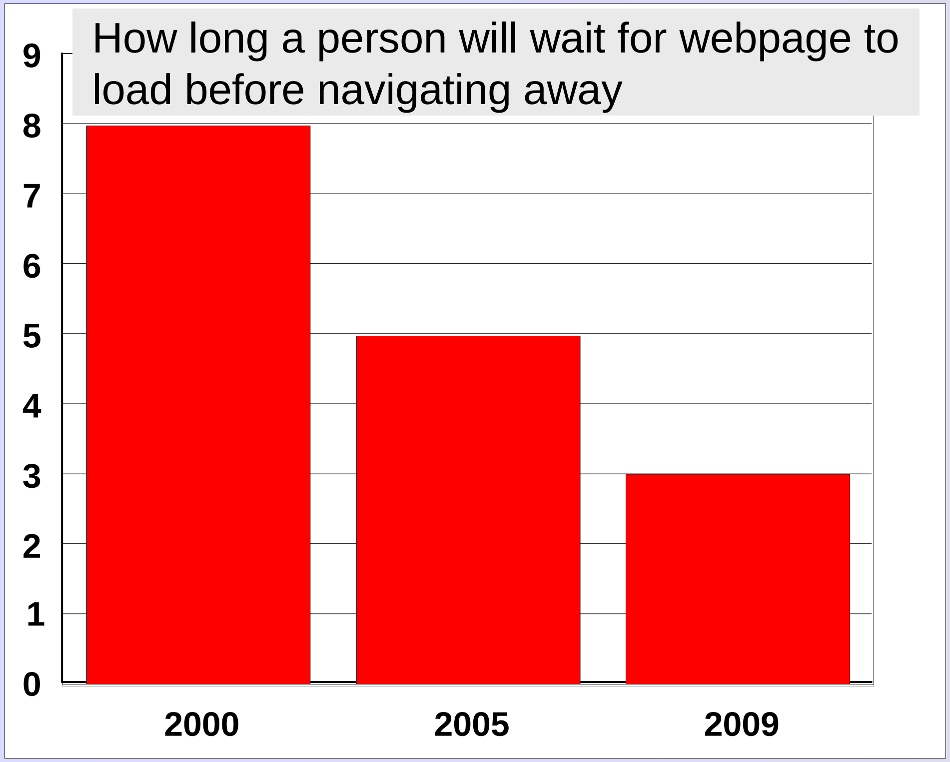
The YouTube Video Editor lets you do basic clip editing and also lets you swap the audio for a selection of music tracks.

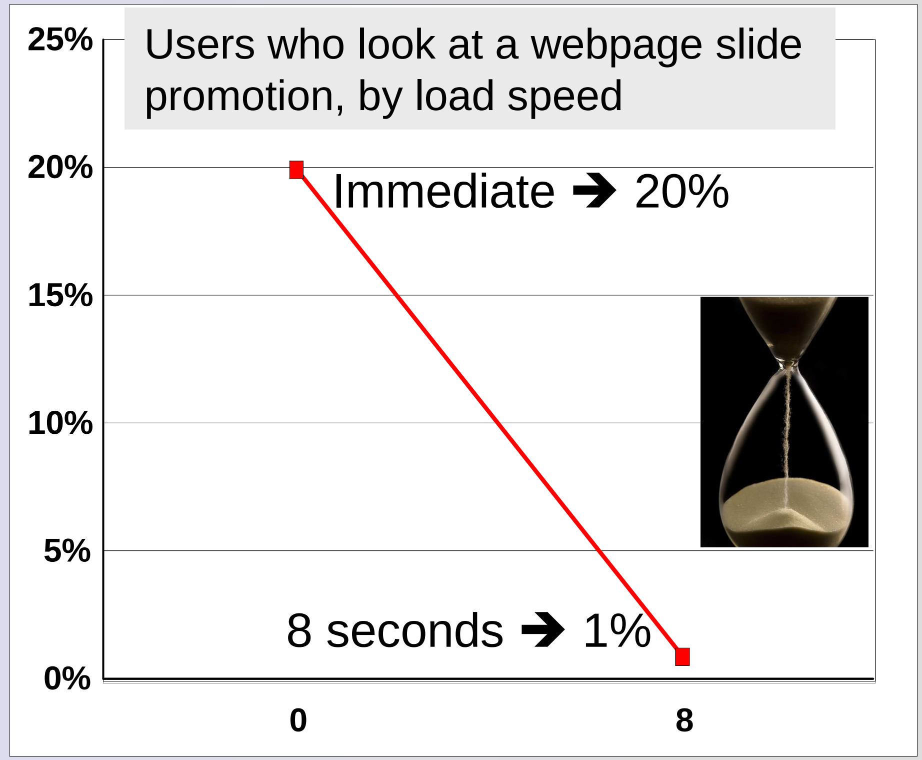
Unfortunately, it's painfully limited and slow – to the point it's hard to imagine doing much more than trimming videos with it.

- **Corel VideoStudio**. Reviewed by: CNET Staff on February 27, 2009.
- Except for one drawback, Corel VideoStudio is an outstanding video creator and editor.
- Its main flaw is its lack of speed.
  - It installs slowly.
  - It loads slowly.
  - It works slowly.

Caveat: I have never used these products and neither endorse nor disparage their use.

Slow Webpage Load Times



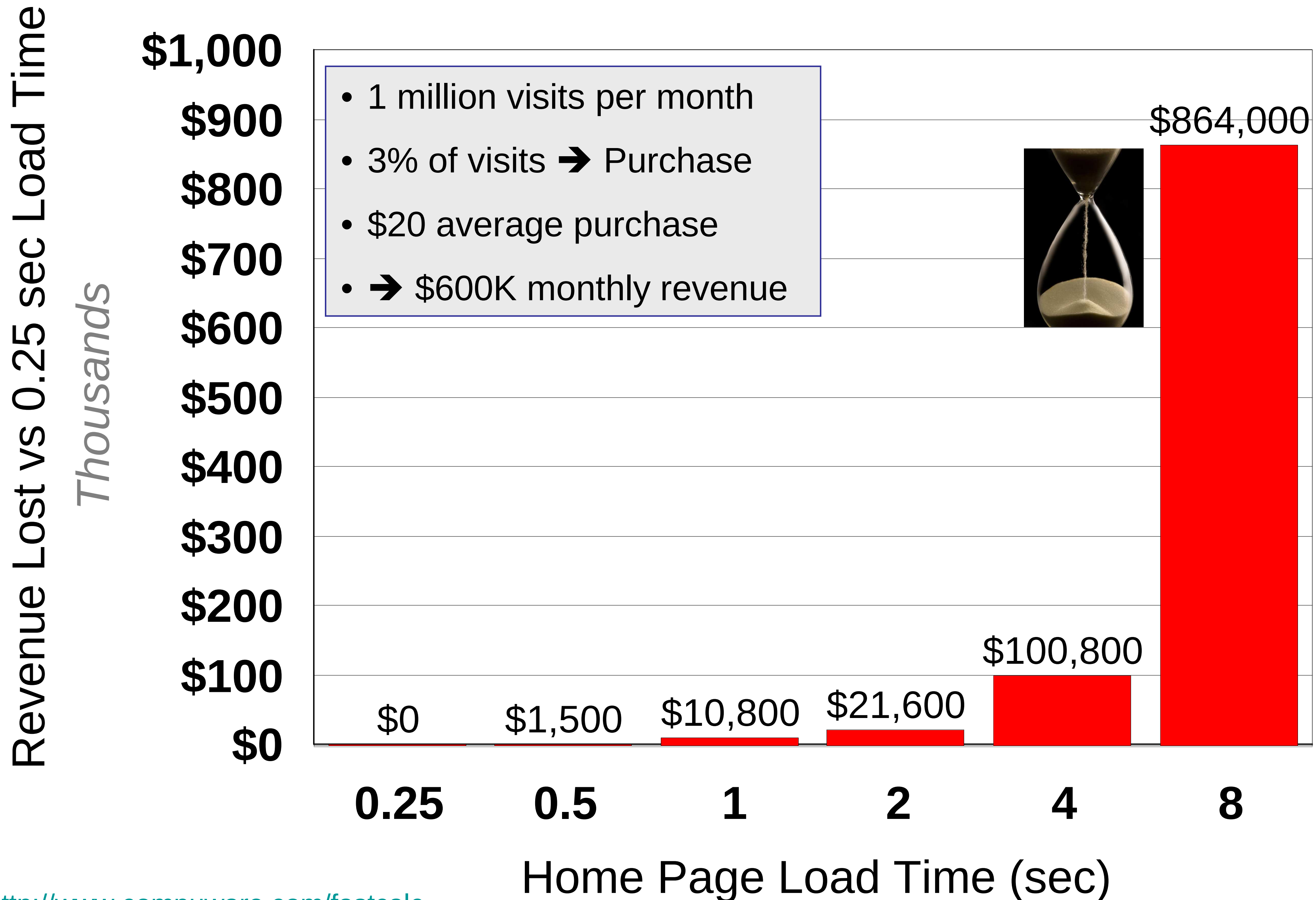


52% of online shoppers say quick page loading is important to their site loyalty.

2009 Forester, Nielsen Norma, and Akamai Studies, Technology Review

http://www.gamingindustrywire.com/article41142.html

Slow Webpage Load Times



http://www.compuware.com/fastcalc

Optimizing Webpage Load Time

- Faster fiber
- Higher processor frequency?
- Co-locating all data on webpage
  - Same datacenter
- Fewer things on webpage
- Simpler things on webpage

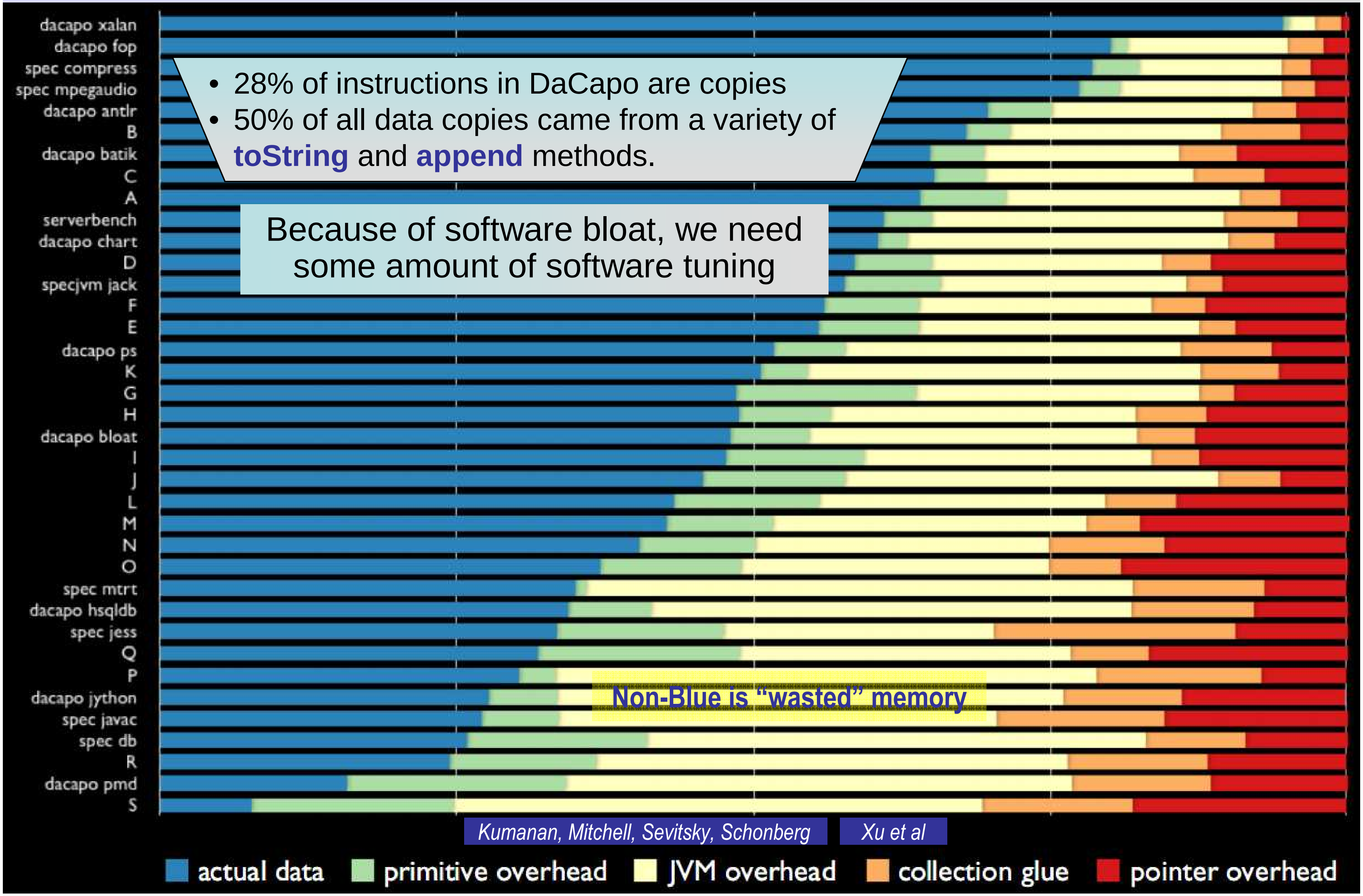
Reduce memory footprint

Issues magnified for smart phones

IBM

High Percentage of “Wasted” Memory in Many Workloads

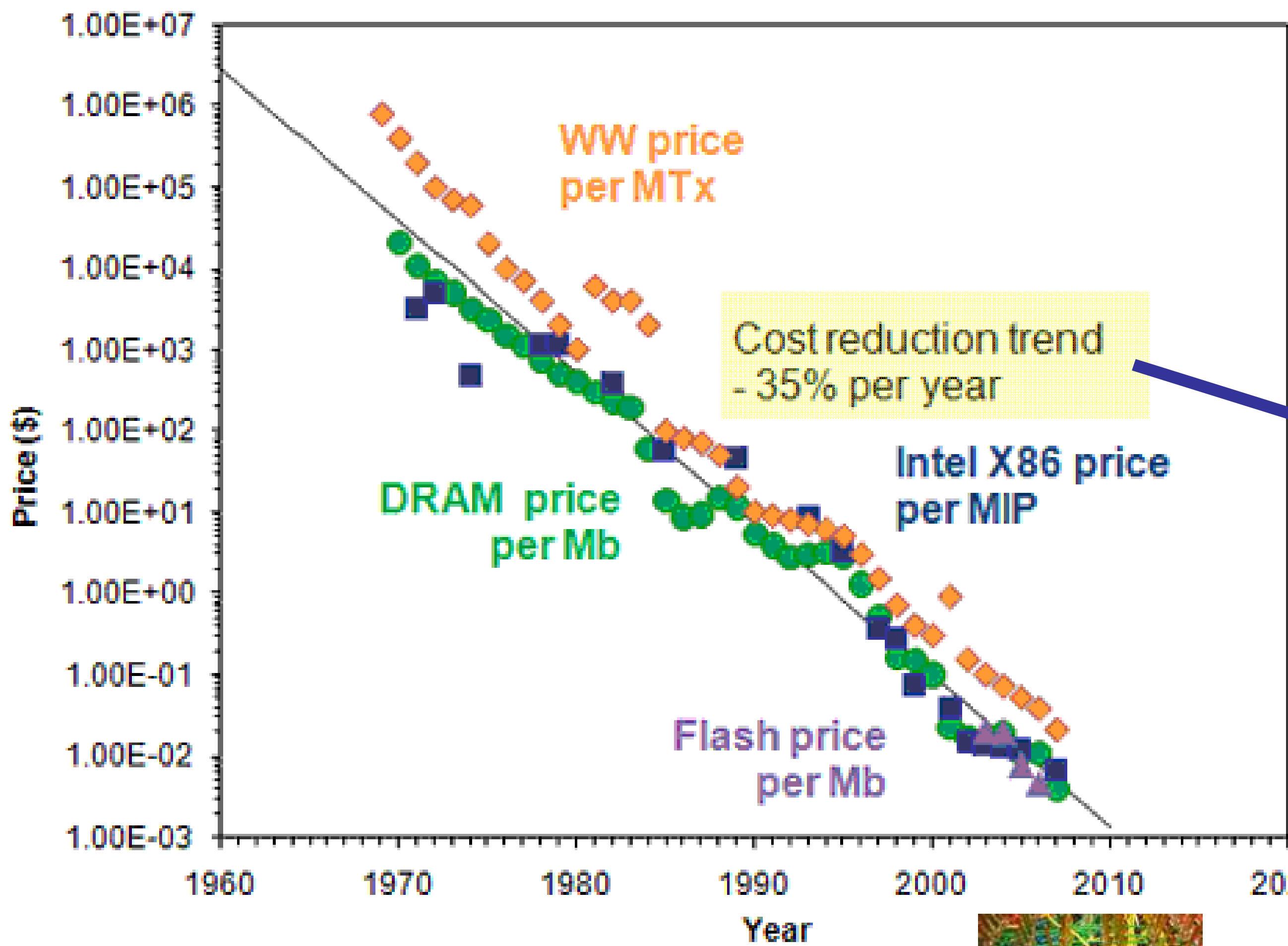
Including large, commercial software



- 28% of instructions in DaCapo are copies
- 50% of all data copies came from a variety of **toString** and **append** methods.

Because of software bloat, we need some amount of software tuning

Prices Since 1970



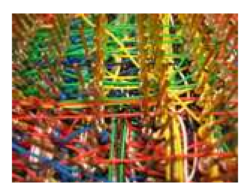
- Million transistors
- Million DRAM bits
- Million Flash bits
- Million Intel Instructions / second

$$\frac{\text{MIPS Price}}{\text{DRAM Price}} = \text{Constant}$$

DRAM is growing part of system cost ➔ *DRAM demand growing faster than MIPS demand*

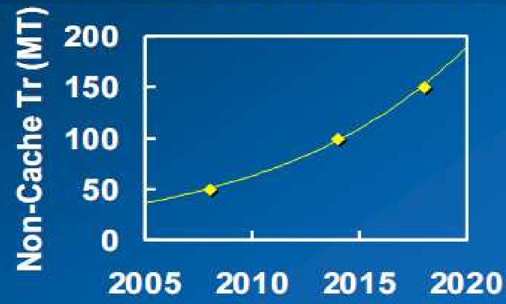
- *Webpages*
- *Java*
- *Video Workloads*
- *Virtualization*

Memory is part of the mess



Memory Mess

- **Implication:** Memory wall coming down
  - Increasing ratio of memory / compute
- More scope for code optimization and VLIW



Non-memory transistors increase only 3X in 10 years

- That's all you can afford (Power)

Memory integration capacity will outpace logic > 10X

- Much more than what is needed

No incentive for constant die size—will decrease?

Why scale the technology if you cannot use it?

Shekhar Borkar

Asia Academic Forum 2010
Nov 10-11, 2010
Ho Chi Minh City, Viet Nam

New Languages for New Workloads

Memory is not the only performance problem.

- Historically, new languages are used for each major new computing task
  - **Fortran:** HPC
  - **C:** OS, Database
  - **Java:** App Servers
  - **Scripting:** Web and Mashups
- ➔ Hard to optimize across tiers developed at different times
  - Database
  - App Server
  - Web Server

Complexity is part of the mess

- Frequency slowdown means we have to do more merging
  - Can't just compose separate apps the way we did in the past
- Hard work:
  - Need insight
  - Need tools
  - Need languages and programming models
- Starting from scratch attractive
  - e.g. Amazon, EBay, Google, Facebook
- But expensive and not always possible
  - Even startups need some inter-operability, eg. credit card authentication

IBM

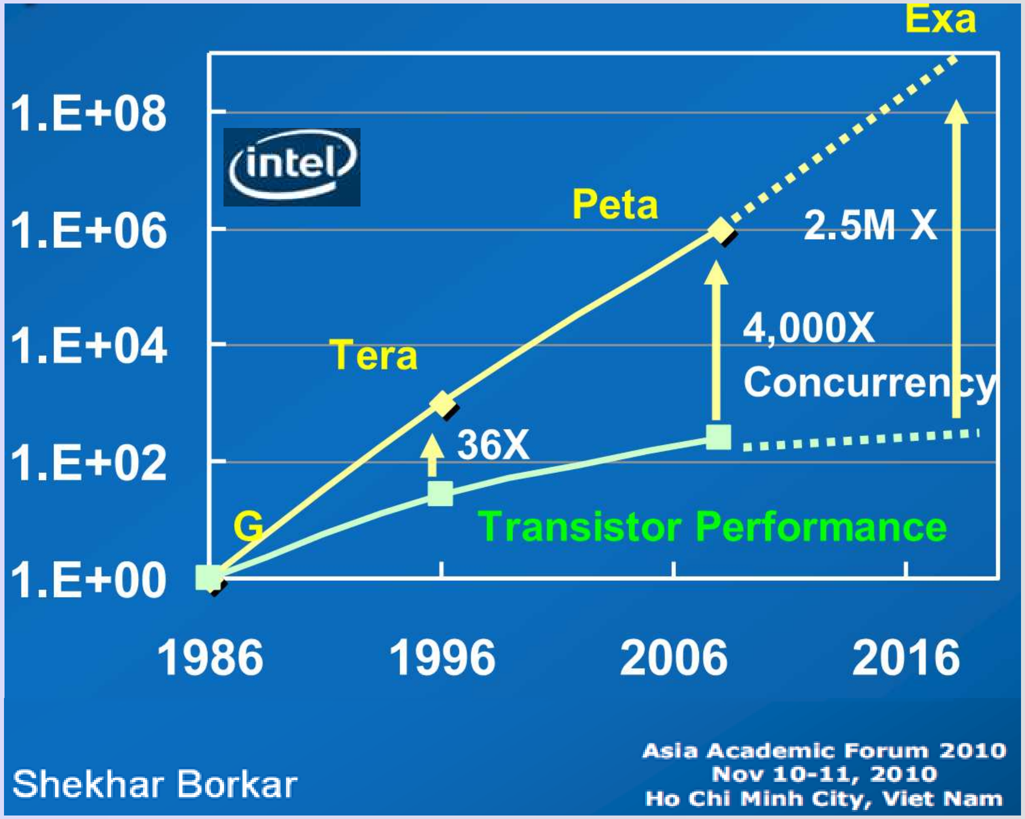
Insight, Tools, and Languages

Start with tools to give insight

Philosophy: Gradual Path to Parallelism

– Write multi-threaded code under assumption of 2-way

  - Improve (over time) as need more parallelism for performance



Optimizing Webpage Load Time

- Faster fiber
- Higher processor frequency?
- Co-locating all data on page
  - Same datacenter
- Fewer things on page
- Simpler things on page

How do I know where to start?

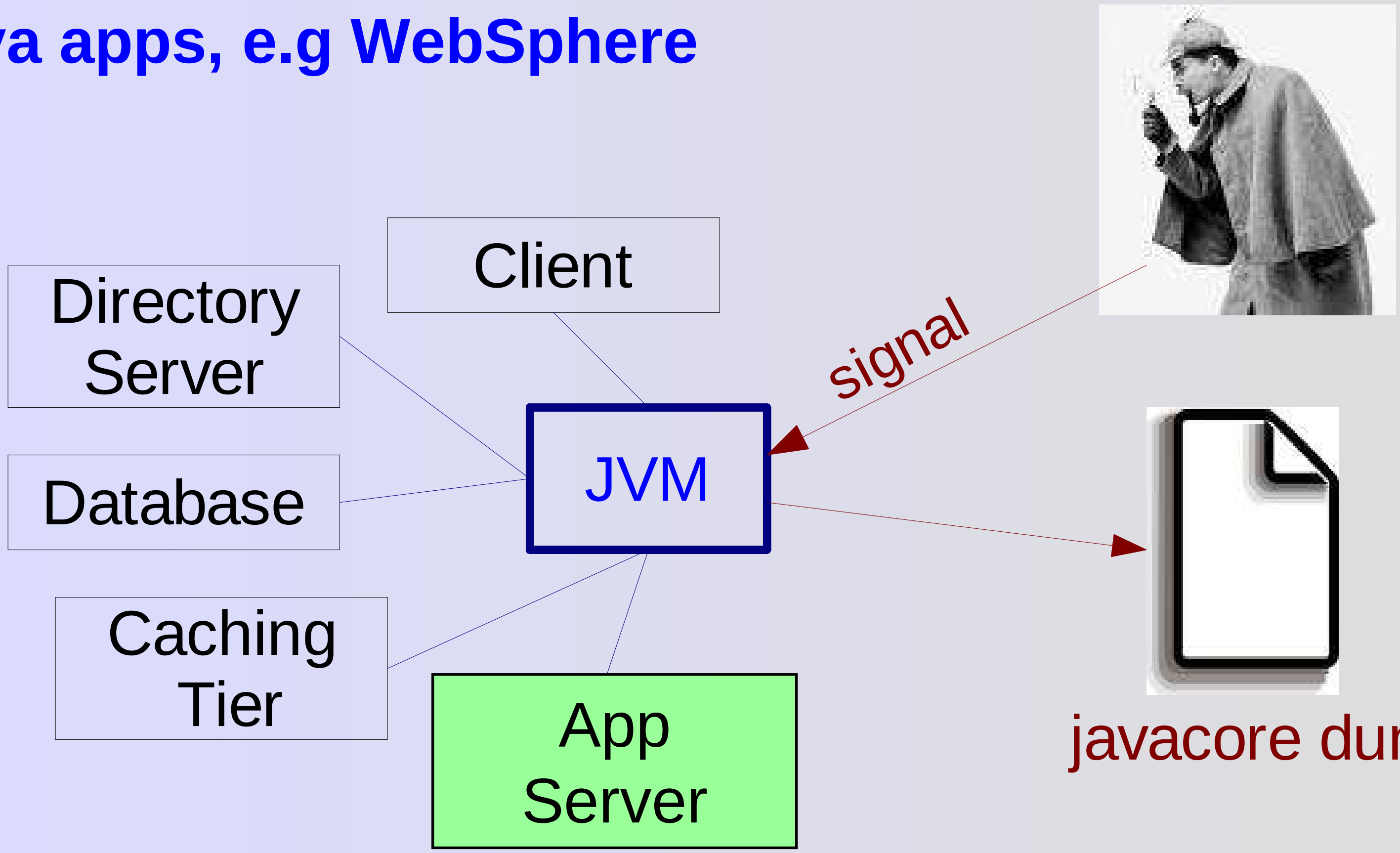
Production Deployment Constraints

- *Production Deployment Constraints*

- *Recompile the application?* ***NO!***
- *Instrument the application?* ***NON!***
- *Deploy a fancy monitoring agent?* ***NEIN!***
- *Analyze the source code?* ***ノー!***
- *Perturb the running system?* ***yIntagh !***

Clues Available

- *Basic operating system utilities (e.g.* **`ps`**, **`vmstat`***)*
- *Log files*
- **Java apps, e.g WebSphere**



Sample Javacore Fragment

```
2LKREGMON        VM mem segment list lock (0x00324CD0): <unowned>
2LKREGMON        MM_CopyScanCacheList::cache lock (0x00324D28): <unowned>
2LKREGMON        MM_CopyScanCacheList::cache lock (0x00324D80): <unowned>
2LKREGMON        FinalizeListManager lock (0x00324DD8): <unowned>
2LKREGMON        Thread public flags mutex lock (0x00324E30): <unowned>
2LKREGMON        Thread public flags mutex lock (0x00324E88): <unowned>
2LKREGMON        &(slaveData->monitor) lock (0x00324EE0): <unowned>
3LKNOTIFYQ         Waiting to be notified:
3LKWAITNOTIFY          "Finalizer thread" (0x414B1B00)
2LKREGMON        Thread public flags mutex lock (0x00324F38): <unowned>
2LKREGMON        Thread public flags mutex lock (0x00325040): <unowned>
2LKREGMON        Thread public flags mutex lock (0x00325098): <unowned>
…
ULL
NULL         ------------------------------------------------------------------------
0SECTION       THREADS subcomponent dump routine
NULL         =================================
NULL
1XMCURTHDINFO  Current Thread Details
NULL         ----------------------
….
3XMTHREADINFO      "Uncle Egad's VP Sender 2" (TID:0x47C4EF00, sys_thread_t:0x4C451C60, state:CW, native ID:0x00001160) prio=5
4XESTACKTRACE         at java/lang/Object.wait(Native Method)
4XESTACKTRACE         at java/lang/Object.wait(Bytecode PC:3)
4XESTACKTRACE         at com/lotus/sametime/core/util/connection/Sender.run(Bytecode PC:44)
4XESTACKTRACE         at java/lang/Thread.run(Bytecode PC:13)
3XMTHREADINFO      "Worker-27" (TID:0x47C4F300, sys_thread_t:0x4C452108, state:CW, native ID:0x000013E8) prio=5
4XESTACKTRACE         at java/lang/Object.wait(Native Method)
4XESTACKTRACE         at java/lang/Object.wait(Bytecode PC:3)
4XESTACKTRACE         at org/eclipse/core/internal/jobs/WorkerPool.sleep(Bytecode PC:52)
4XESTACKTRACE         at org/eclipse/core/internal/jobs/WorkerPool.startJob(Bytecode PC:77)
4XESTACKTRACE         at org/eclipse/core/internal/jobs/Worker.run(Bytecode PC:223)
NULL         ------------------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL         =================================
1CLTEXTCLLOS     Classloader summaries
1CLTEXTCLLSS                        12345678: 1=primordial,2=extension,3=shareable,4=middleware,5=system,6=trusted,7=application,8=delegating
2CLTEXTCLLOADER                                     p---st-- Loader *System*(0x004768A8)
3CLNMBRLOADEDLIB                                    Number of loaded libraries 4
3CLNMBRLOADEDCL                                                  Number of loaded classes 1374
2CLTEXTCLLOADER                                     -x--st-- Loader com/ibm/oti/vm/URLExtensionClassLoader(0x00479428), Parent *none*(0x00000000)
3CLNMBRLOADEDLIB                                    Number of loaded libraries 0
3CLNMBRLOADEDCL                                                  Number of loaded classes 50
2CLTEXTCLLOADER                                     -----ta- Loader com/ibm/oti/vm/URLAppClassLoader(0x004769C8), Parent com/ibm/oti/vm/URLExtensionClassLoader
```

Clues → WAIT Tool

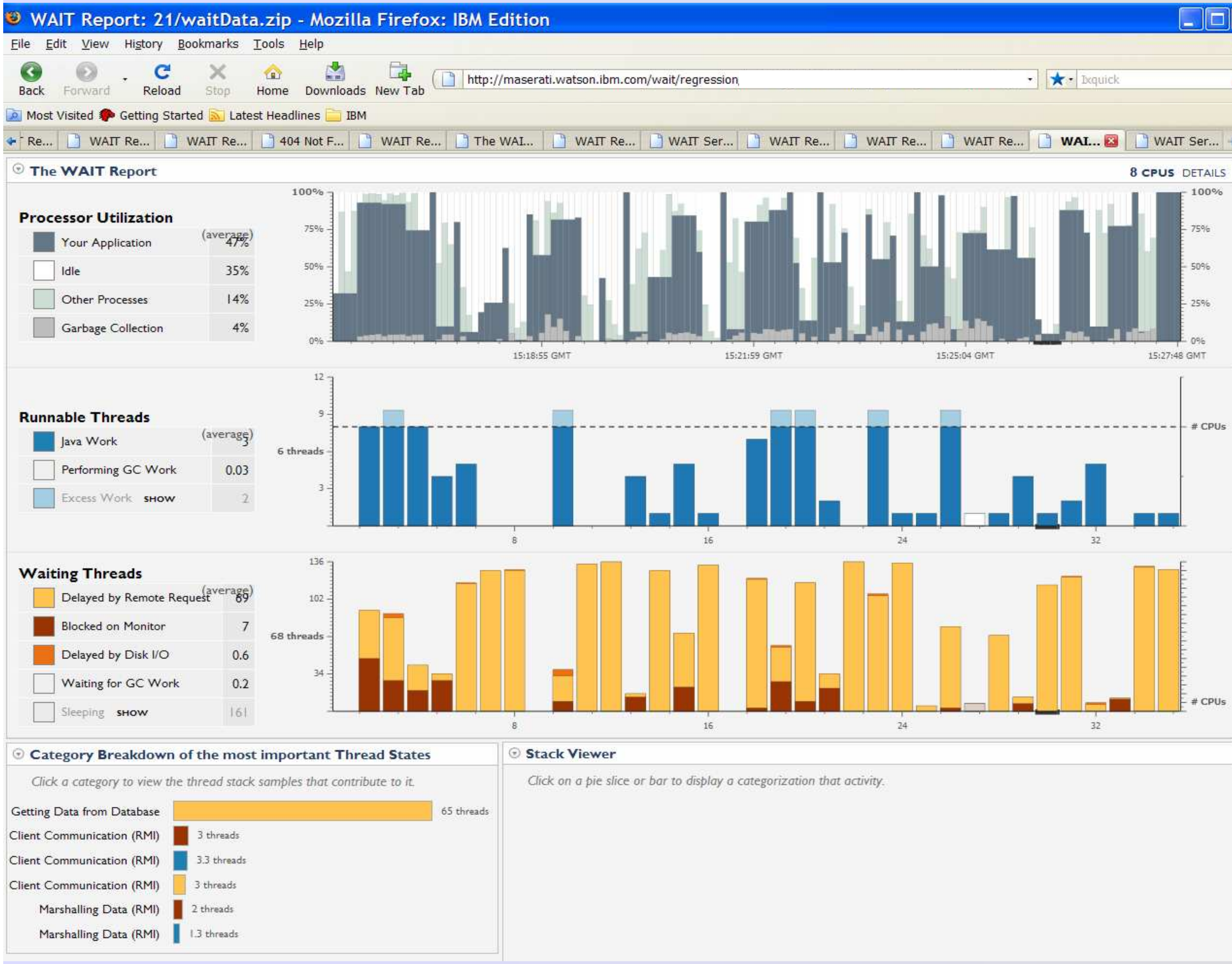
- **WAIT** uses expert rules to interpret data
- **WAIT** focuses on primary bottlenecks
  – Gives high-level, whole-system, summary of performance inhibitors
- **WAIT** is zero install
  – Leverages built-in data collectors
  – Reports results in a browser

- **WAIT** is non-disruptive
  – No special flags, no restart
  – Use in any customer or development location
- **WAIT** is low-overhead
  – Uses only infrequent samples of an already-running app

- **WAIT** does not capture sensitive user data
  – No source code, personal ID numbers, credit card numbers
- **WAIT** uses centralized knowledge base
  – Allows rules and knowledge base to grow over time

Customer A Customer B

Example WAIT Report



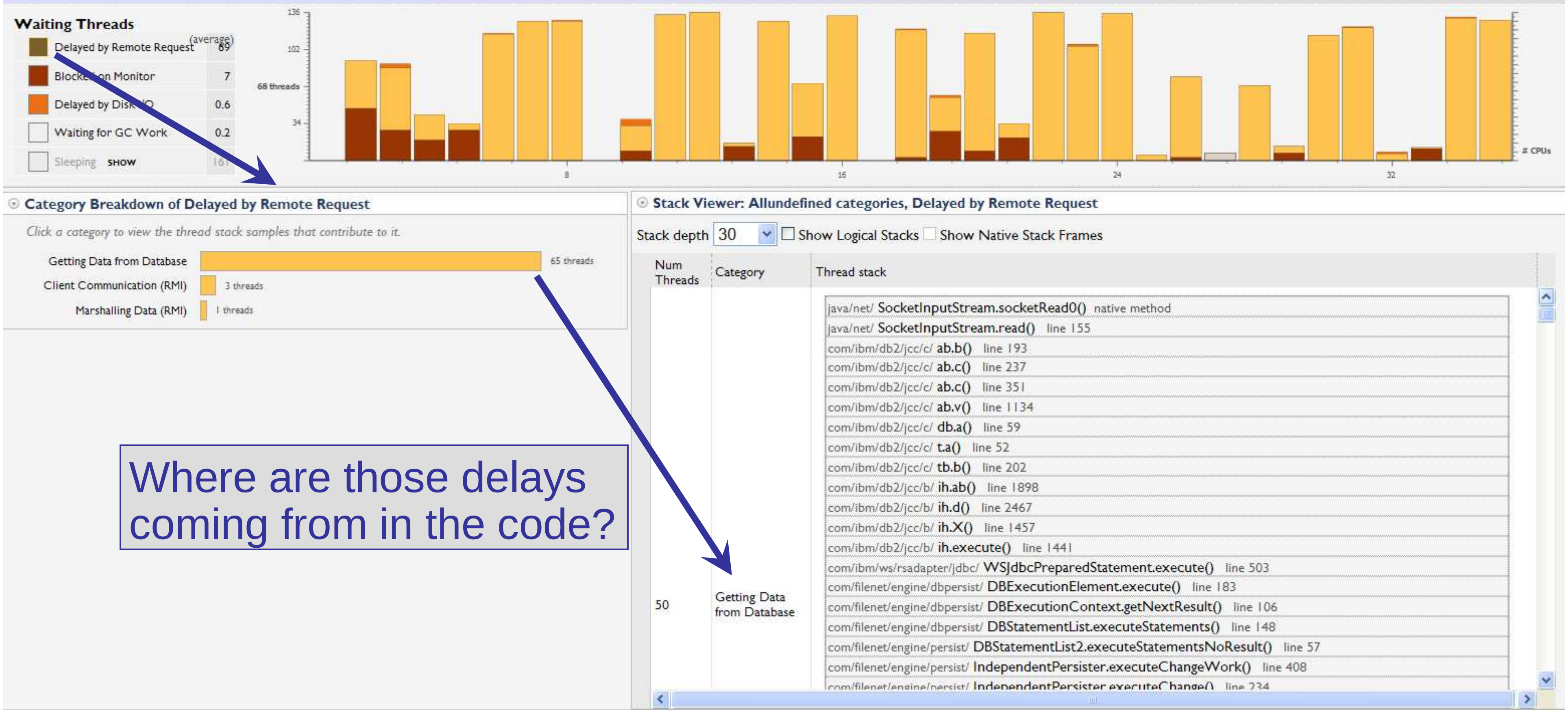
What is the CPU doing?

What Java work is running?

What Java work cannot run?

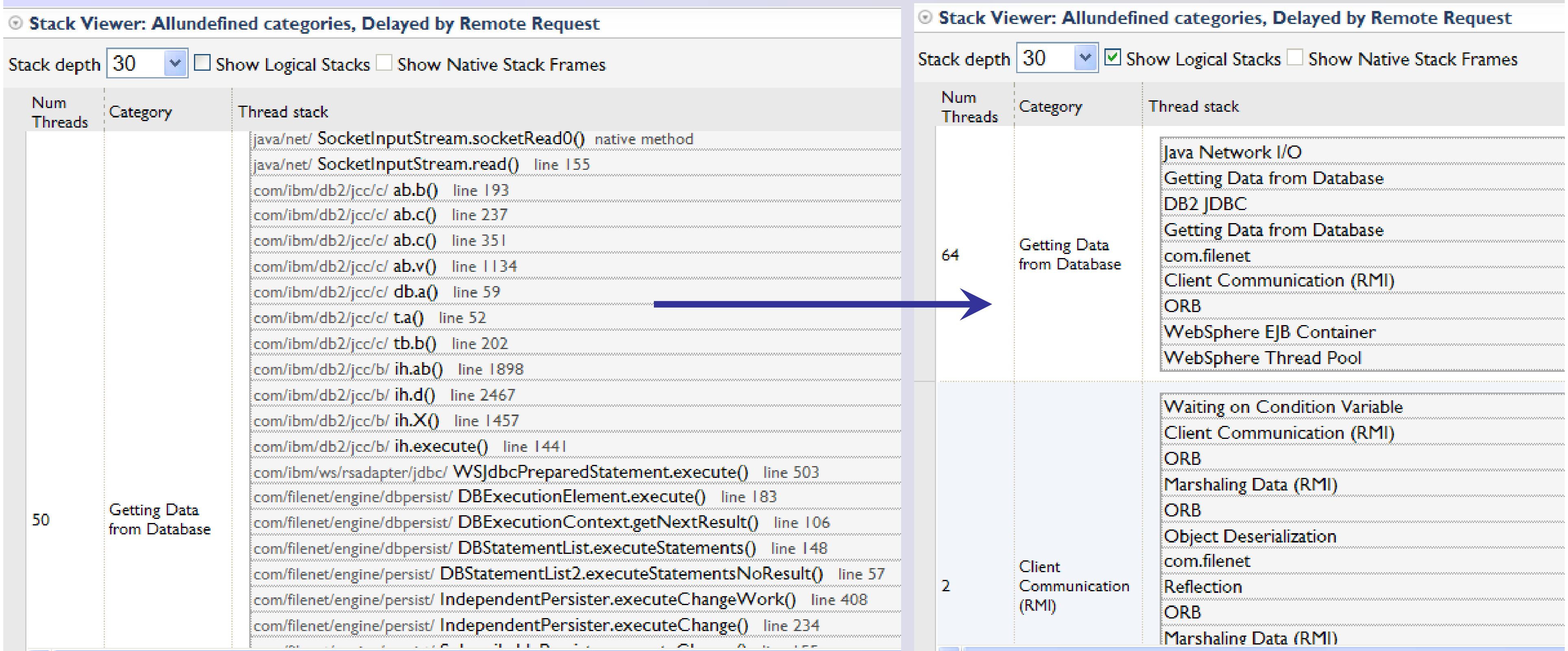
WAIT Report: What is the main cause of delay?

Drill down by clicking on legend item



Where are those delays coming from in the code?

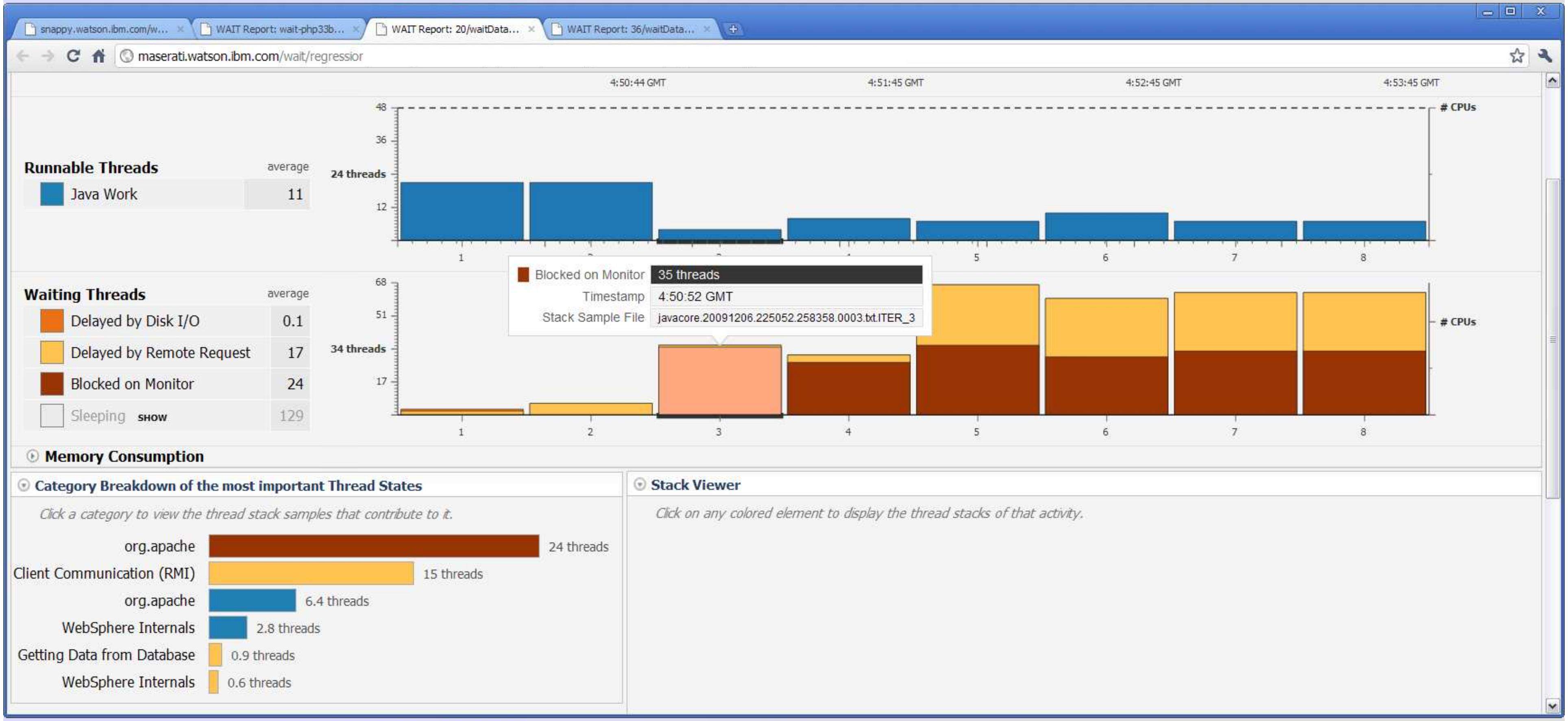
Physical and Logical Stacks



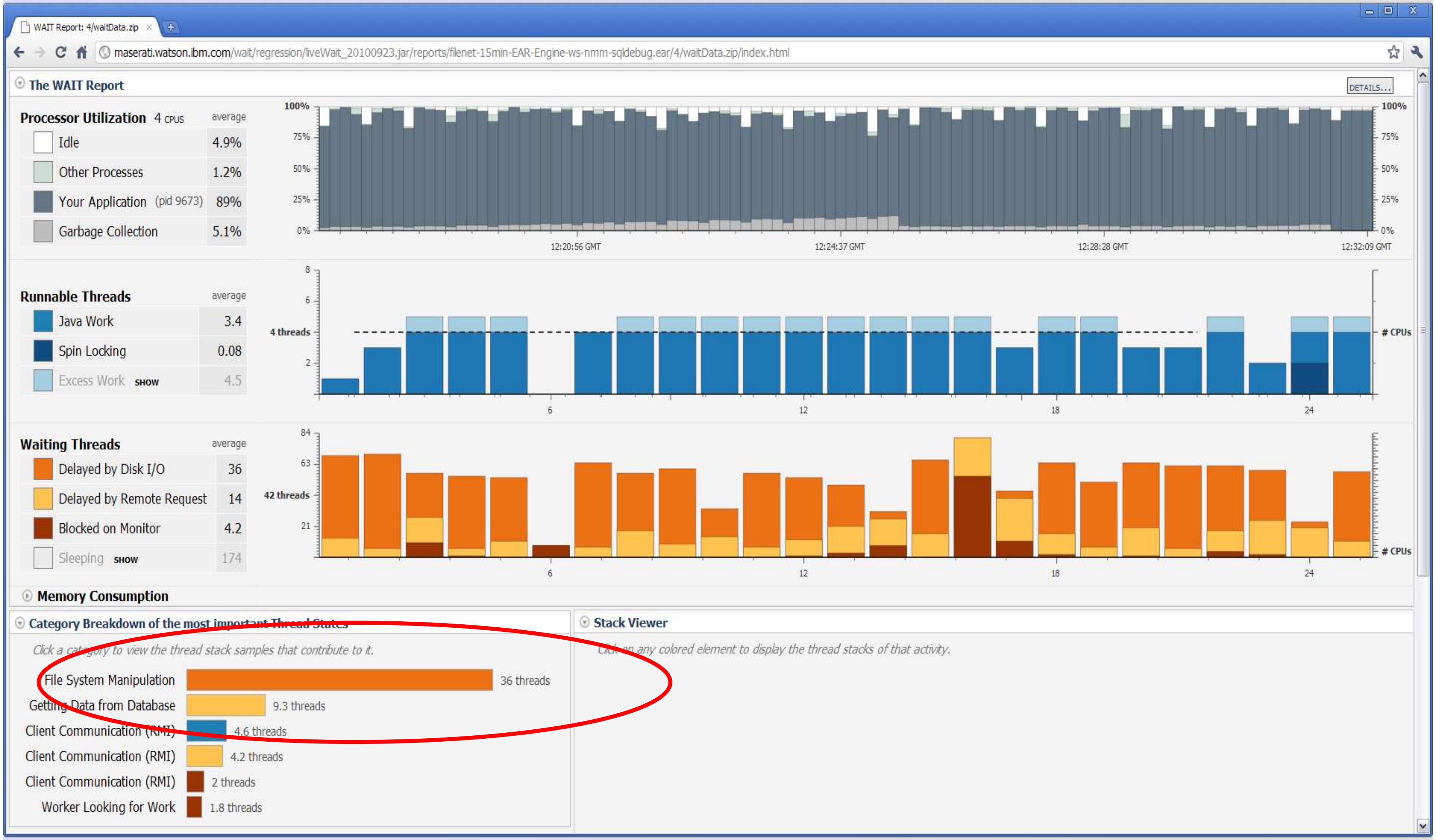
WAIT: Logical view of layers and frameworks

IBM

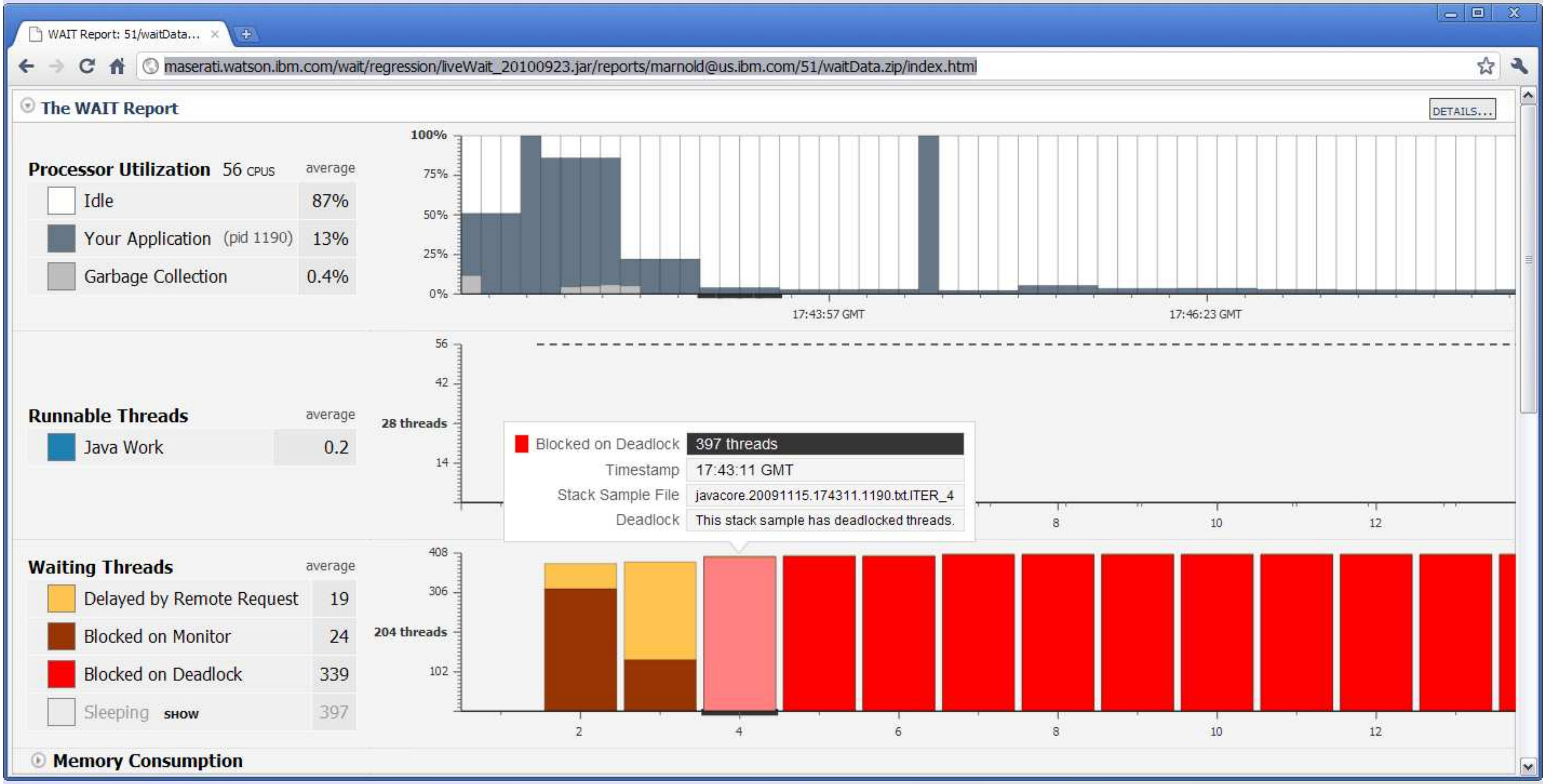
Example Report: Lock Contention



Filesystem Bottleneck

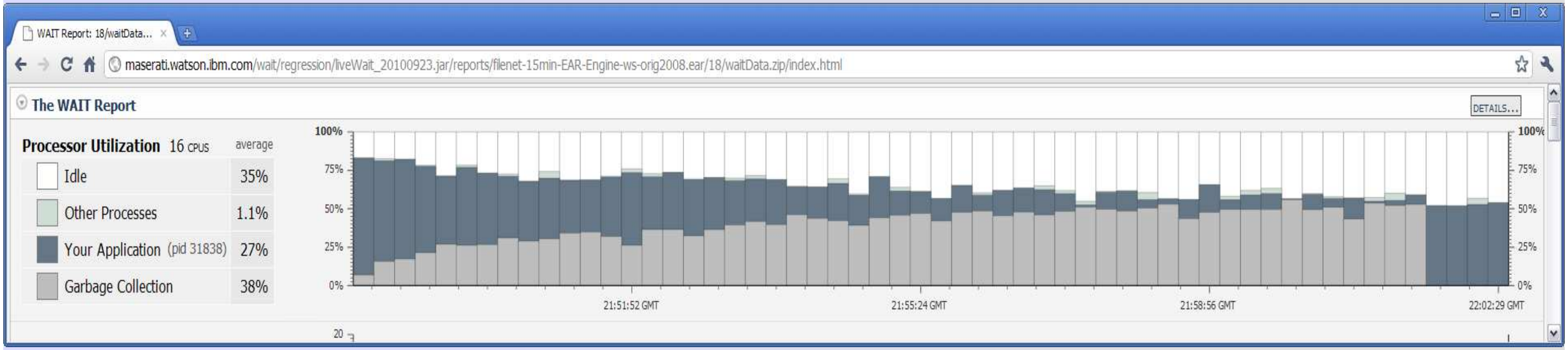


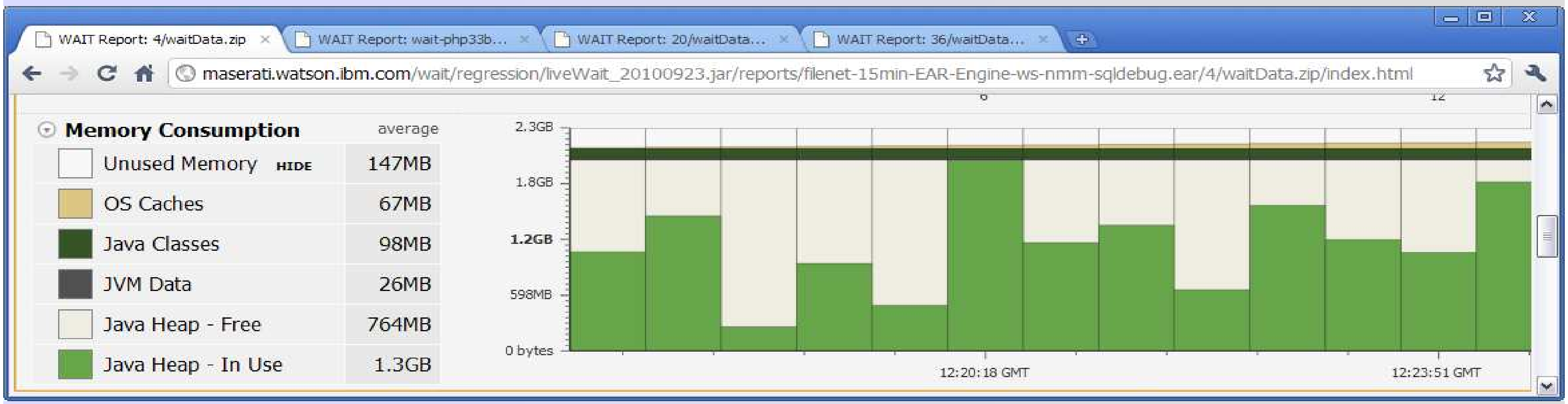
Deadlock



WAIT Report: 51/waitData...
maserati.watson.ibm.com/wait/regression/liveWait_20100923.jar/reports/marnold@us.ibm.com/51/waitData.zip/index.html
The WAIT Report
DETAILS...
Processor Utilization 56 CPUS average
Idle 87%
Your Application (pid 1190) 13%
Garbage Collection 0.4%
100%
75%
50%
25%
0%
17:43:57 GMT
17:46:23 GMT
Runnable Threads average
Java Work 0.2
56
42
28 threads
14
Blocked on Deadlock 397 threads
Timestamp 17:43:11 GMT
Stack Sample File javacore.20091115.174311.1190.txt.ITER_4
Deadlock This stack sample has deadlocked threads.
Waiting Threads average
Delayed by Remote Request 19
Blocked on Monitor 24
Blocked on Deadlock 339
Sleeping SHOW 397
408
306
204 threads
102
2
4
6
8
10
12
Memory Consumption

Memory Analysis





Tooling in Software Lifecycle

WAIT applies everywhere in cycle.
– **Key:** Lightweight and simple

Entry Point

Code & Tune

Refine compiler options/directives
Use optimized libraries
Recode part of application
Introduce/increase parallelism*

Build

Use latest compiler
Turn on optimization
Enable parallelization*

Performance Tuning

Analyze

Static code analysis
Find “hot spots”
Identify performance bottlenecks
Identify scalability bottlenecks*

Test & Debug

Run Application
Check correctness
Check concurrency issues*

Monitor

Measure performance
Collect execution stats
Validate performance gains
Gather stats on scalability*

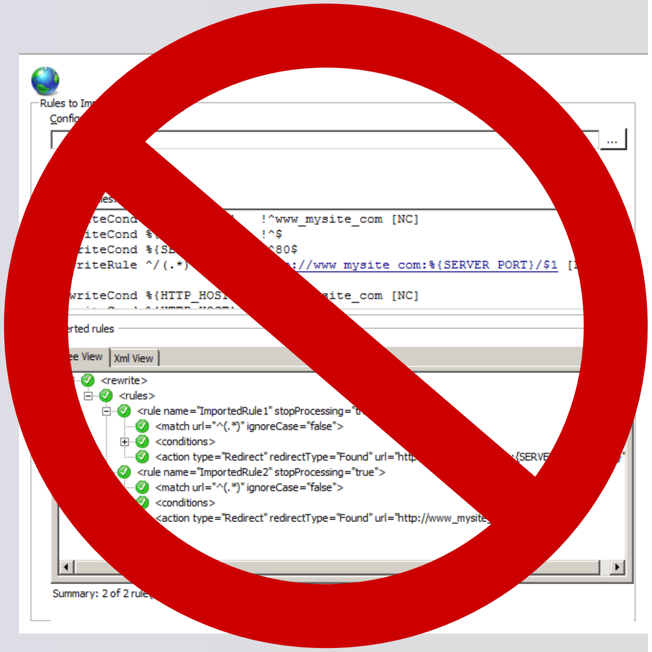
Exit Point

Entry Point

* For parallel code

Tuning ≠ Rewrite from Scratch

- Two in-depth case-studies with WAIT tool ➔
  - 5x performance gain
  - 60x performance gain
- Both cases:
  - 30 sets of code changes
  - Each change: 10 lines of code



WAIT Summary

Clean the mess

- WAIT enables high-level, end-to-end optimization of the mess
  - Focus on identifying primary bottleneck
  - Usable with any Java application
    - Large scale or small
  - Similar techniques can be applied to C/C++ and other “native” code
  - Browser interface, agentless, simple to use ➔ Very low barrier to entry

- **Follows philosophy:**
  - Gradually increase parallelism via tuning at each generation

- Lots of opportunities for CGO community:
  - Automate the manual optimizations done using WAIT data, e.g.
    - Better data structures for concurrency
    - Use of concurrent libraries
    - Optimize across tiers, e.g. app server and database

  - **Caveat:** Handle with care. Wholesale static changes often degrade performance.

Limitations of General Purpose CPU

Starting with System 360, we have been lucky to have a general purpose model in computing.

- *But that era may be ending.*
- *Appliance era beginning:*

Key Drivers:

- Need more performance
- Need more performance per watt

Rack | Router | Storage

Gamebox | Desktop | Laptop | Tablet | Cellphone

CPU | GPU | FPGA

What is the new ISA?

- To manage all these things in a common, portable way.

- **Appliance**: Instrument, apparatus, or device for a particular purpose or use.
- **Claim:** To succeed, general purpose products must implement all functions – *including price* – nearly as well as standalone appliances.

→ General purpose is the anomaly

Appliances *vs* General Purpose

Cooking Appliances

- Stove
- Microwave
- Oven
- Toasters

➔ **General purpose failure**

Multi-function Vehicles:

- Car-Boat, Car-Plane, Car-Chair

➔ **General purpose failure**

Knives

- **Appliance:**
  - Butter knife
  - Table knife
  - Carving knife
  - Bread knife
  - Paring knife
- **General Purpose:**
  - Swiss army knife
  - Amazing Ginsu knife

➔ **General purpose failure**

Wristwatch:

- Simple Analog →
- Analog with Date →
- Multi-function Digital →
- Multi-function Digital with Calculator

➔ **General purpose failure**

Claim: To succeed, general purpose products must implement all functions – *including price* – nearly as well as standalone appliances.

→ General purpose is the anomaly

Can we afford the appliance software?

Yes!

	Rack	Router	Storage	
Gamebox	Desktop	Laptop	Tablet	Cellphone
	CPU	GPU	FPGA	

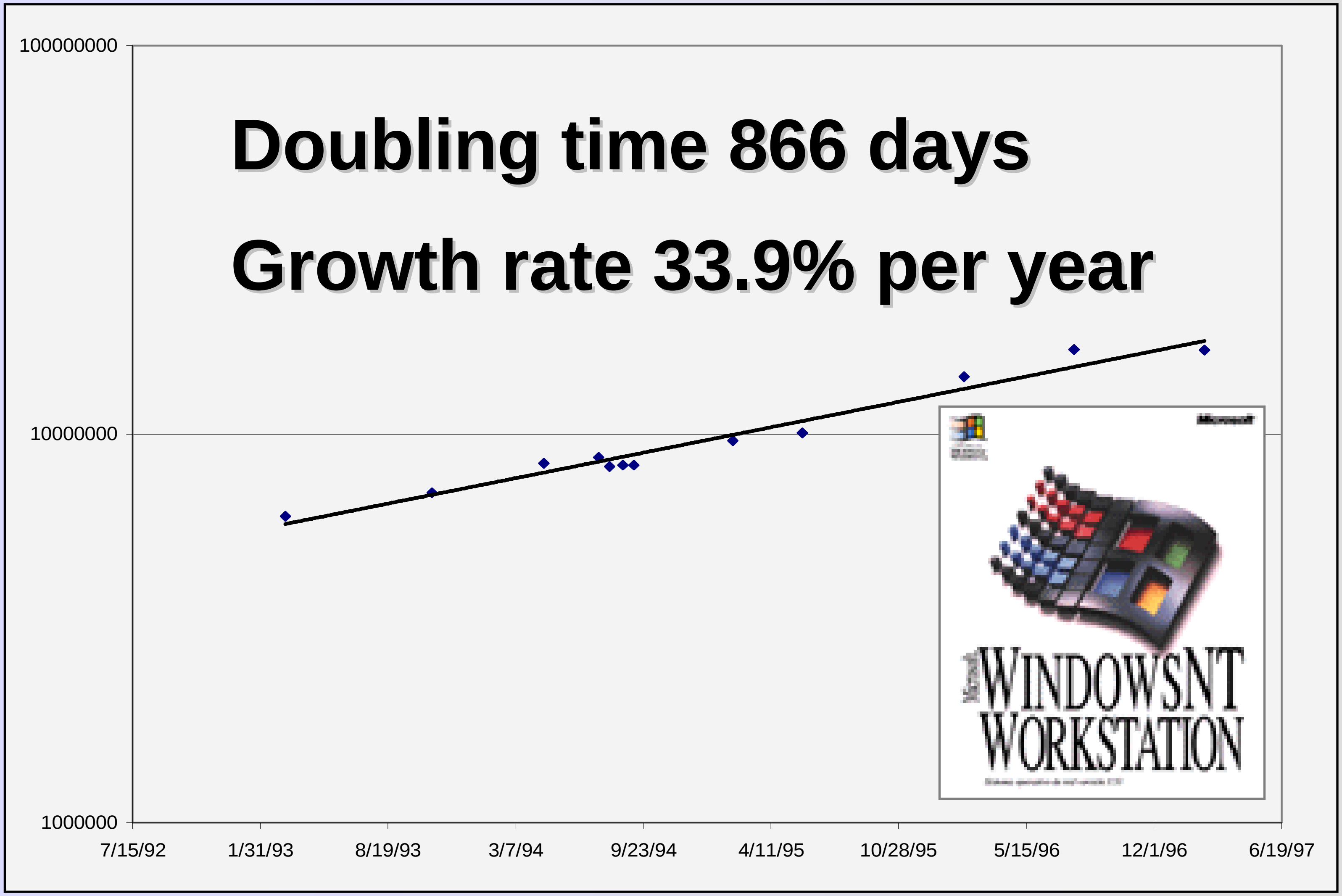
We have to, until there is a new ISA

- Economic / productivity gains from new ISA ➔ There will be attempts.
- *Even in this talk* ☺

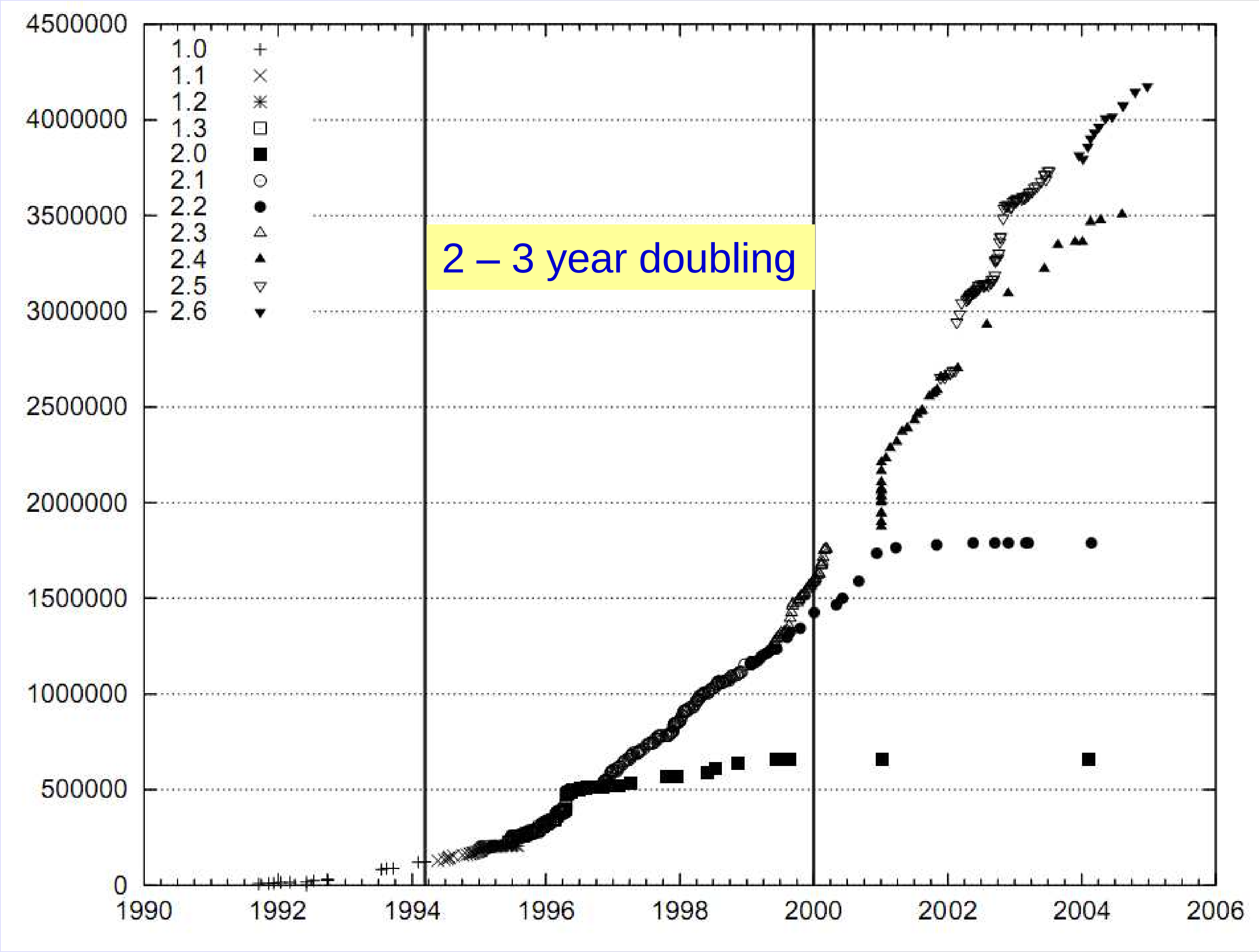
- App store has 400,000 apps in 3 years.

- Software grows exponentially
  - Slower than Moore's Law.
  - But doubling every 0.6 - 6 years.
  - ➔ Equivalent of rewriting all current software over 0.6 - 6 years.

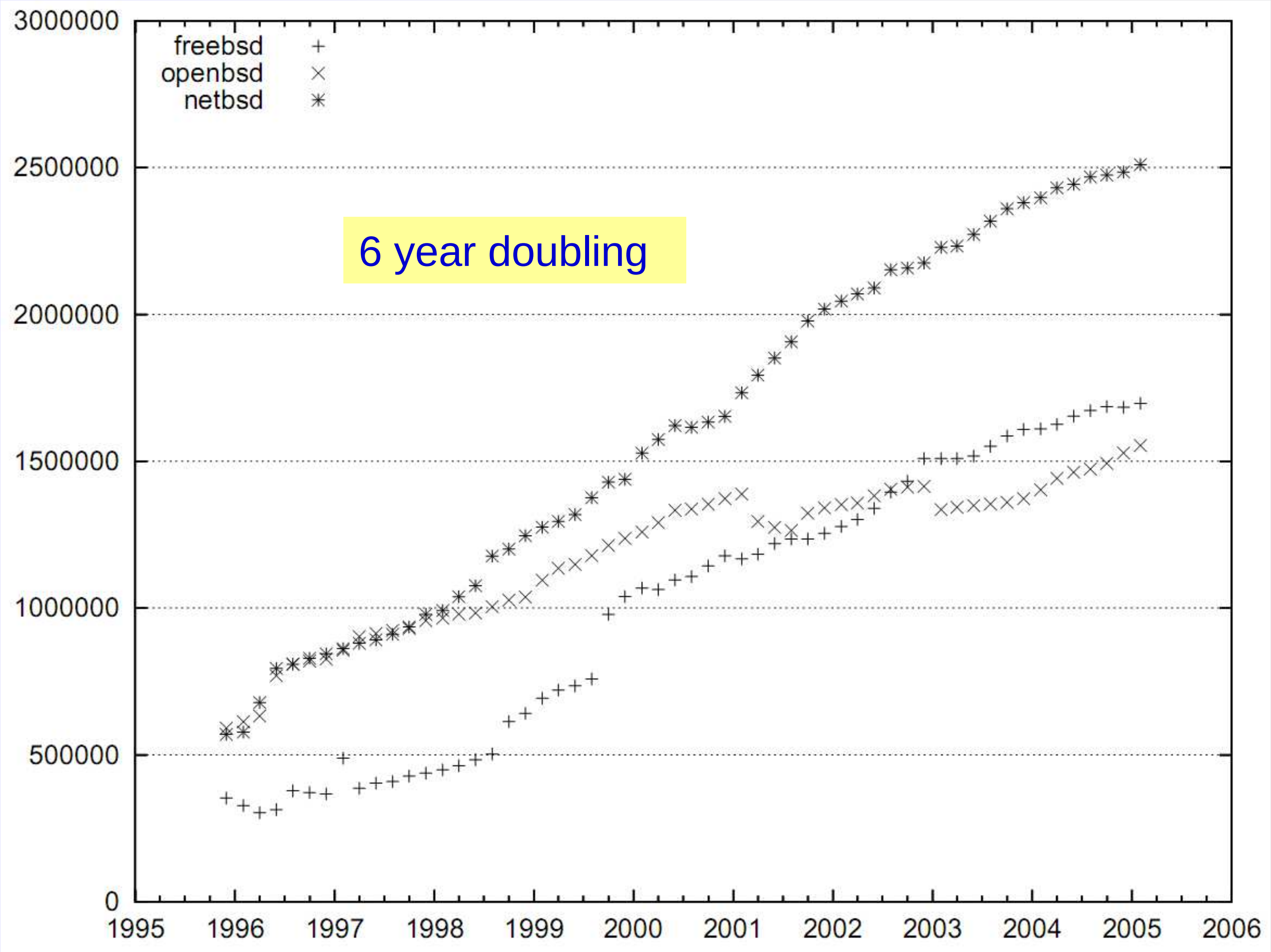
Lines of Code: Windows



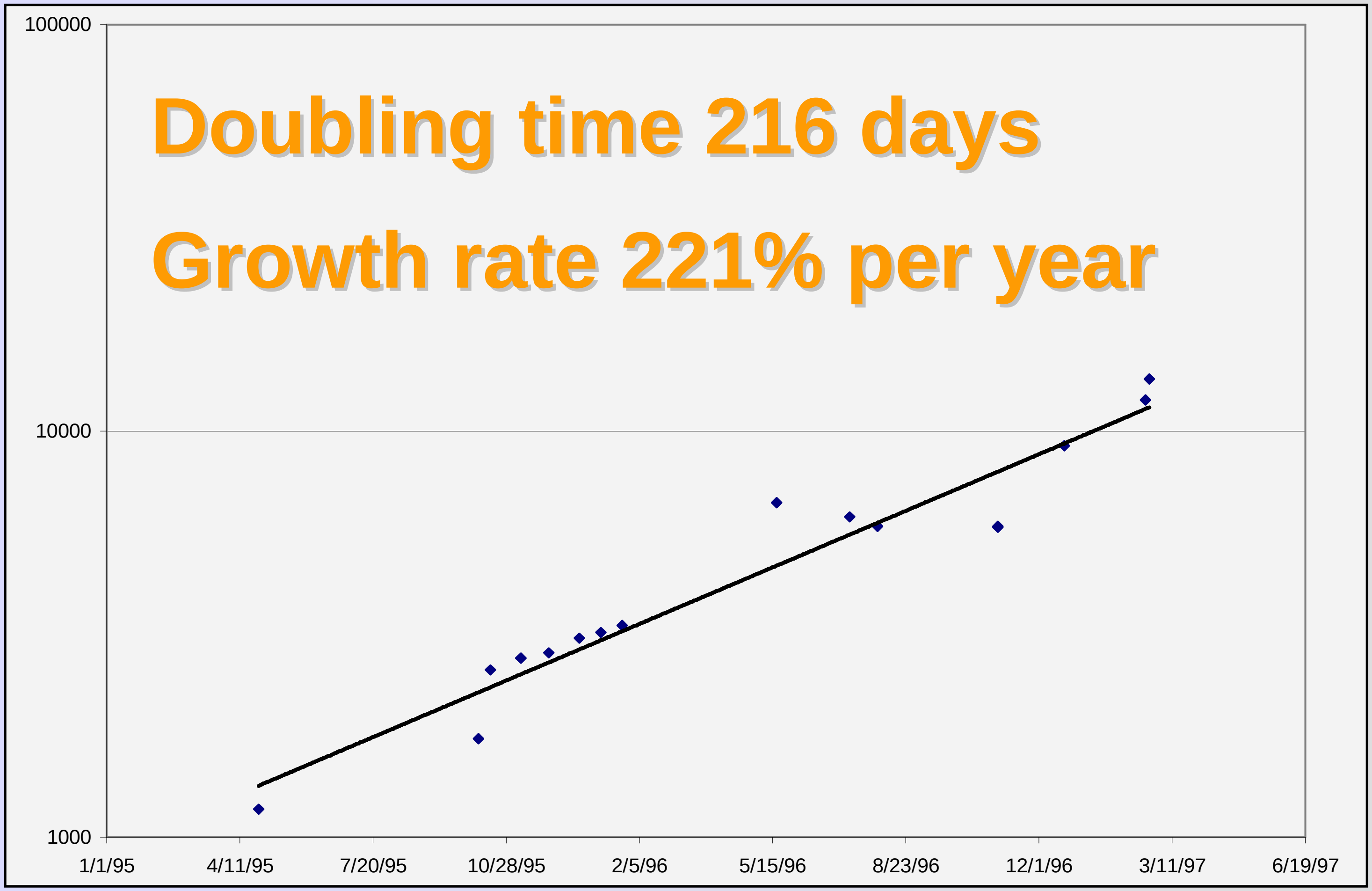
Lines of Code: Linux



Lines of Code: BSD

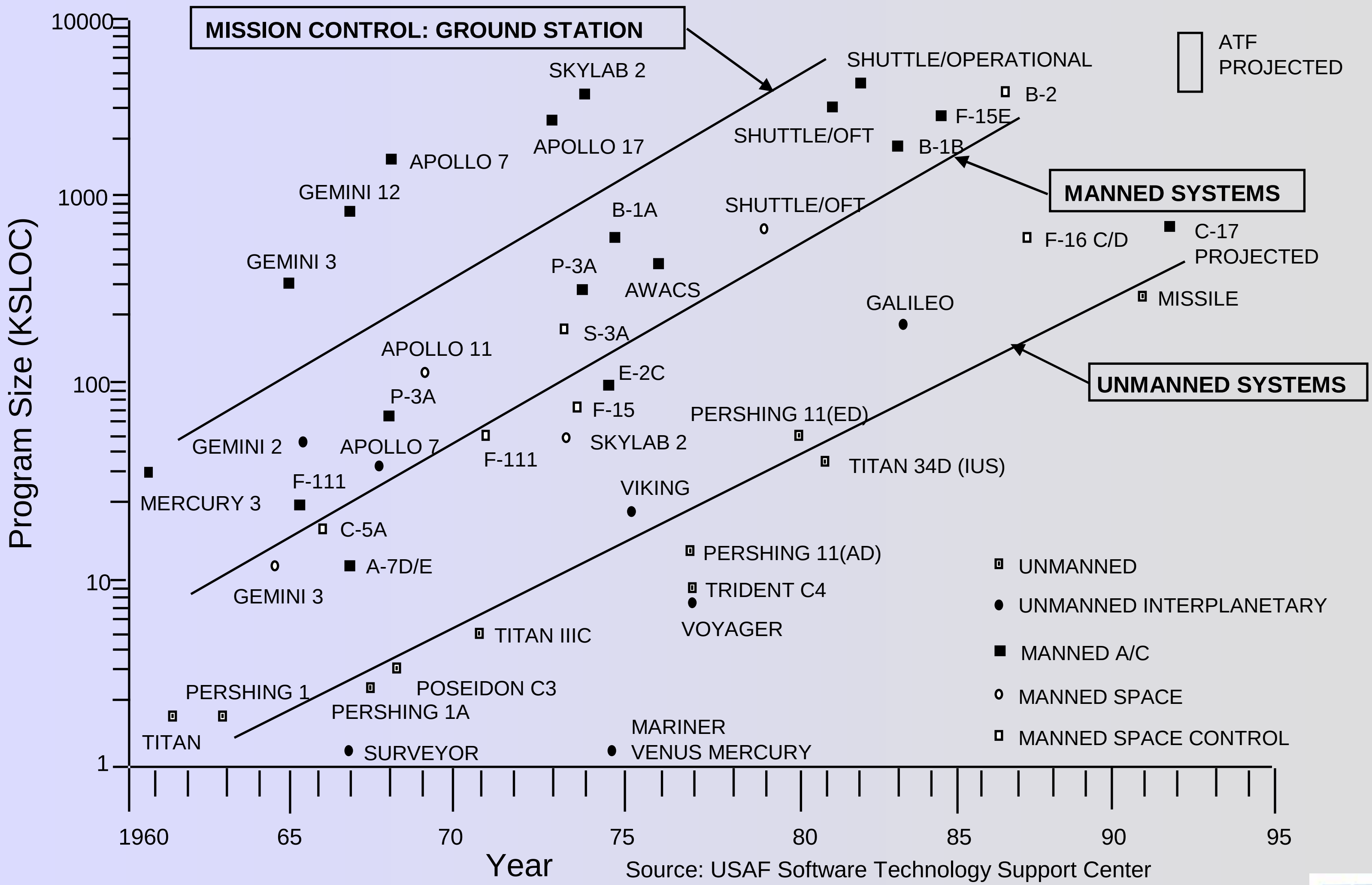


Lines of Code: Browser



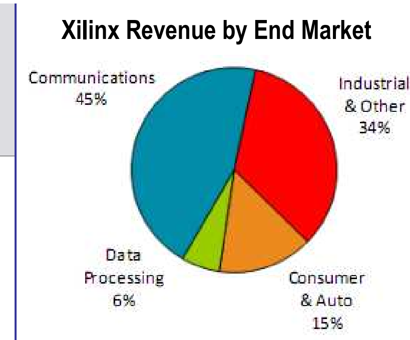
Lines of Code: NASA

2 – 3 year doubling

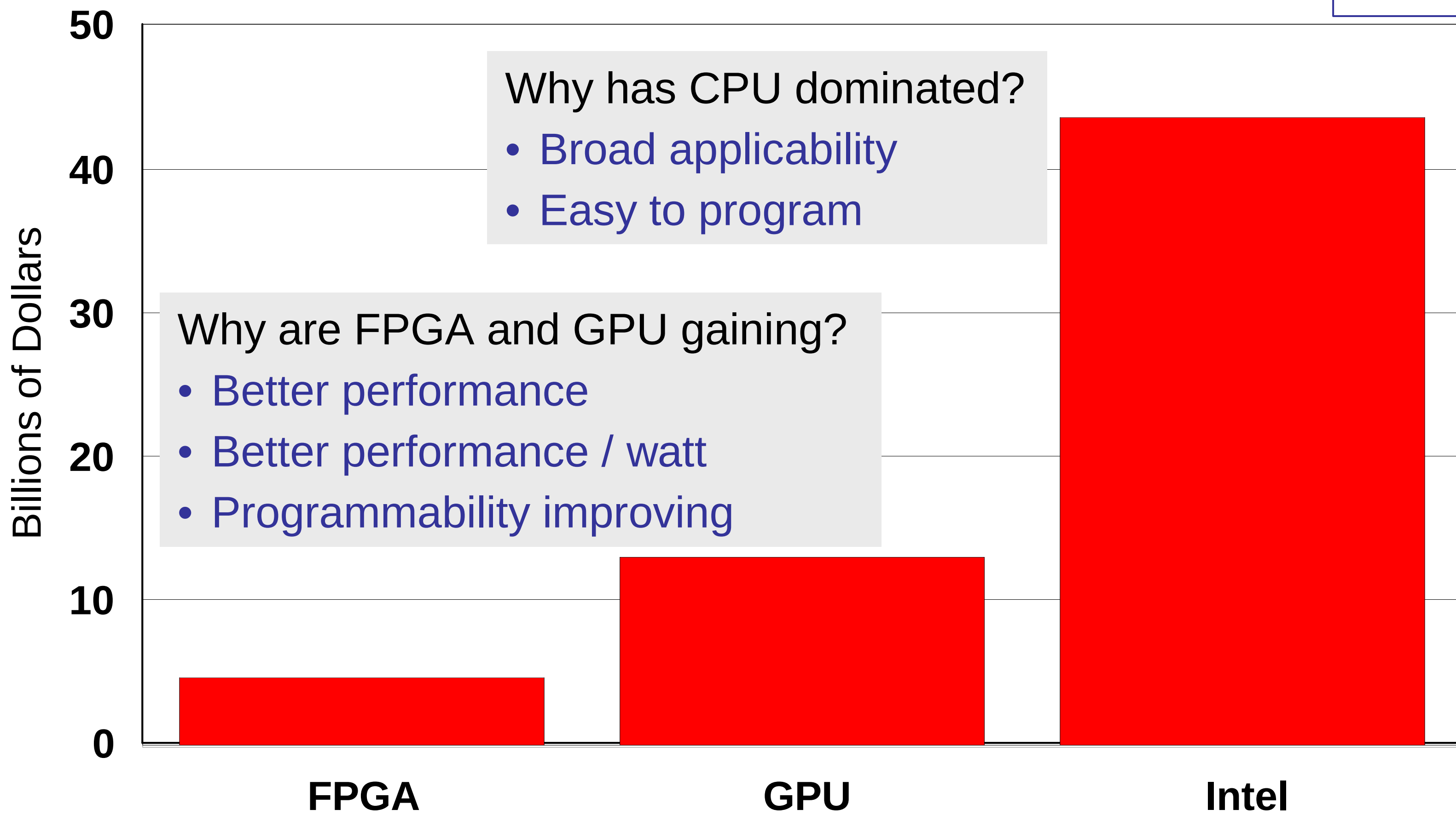


Source: USAF Software Technology Support Center

Computing Devices



Market Size



Why has CPU dominated?
- Broad applicability
- Easy to program

Why are FPGA and GPU gaining?
- Better performance
- Better performance / watt
- Programmability improving

What is the new ISA?
- To manage all computing devices in a common, portable way.

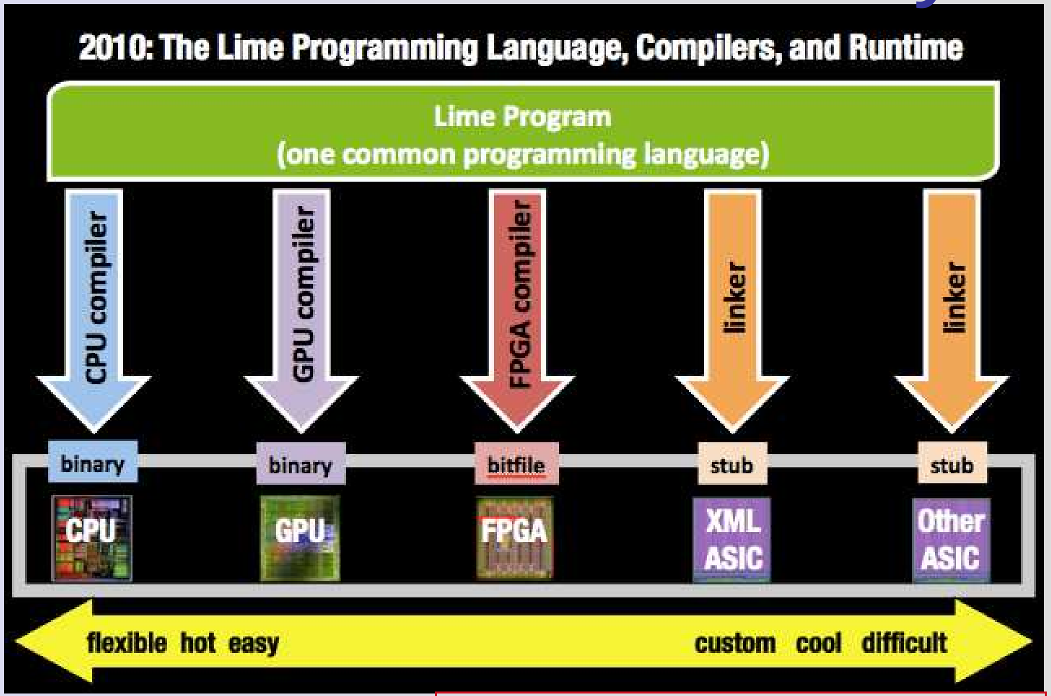
Language for Task

- We tend to develop new languages for each major new computing task:
  - **Fortran:** HPC
  - **C:** OS, Database
  - **Java:** App Servers
  - **Scripting:** Web and Mashups

  - **Lime / Liquid Metal:** FPGAs, GPUs, and CPUs
    - *The new ISA?*

Fixing the mess

Liquid Metal Goal and Vision Summary



GPU	graphics processor
FPGA	field programmable gate array
ASIC	application specific processor

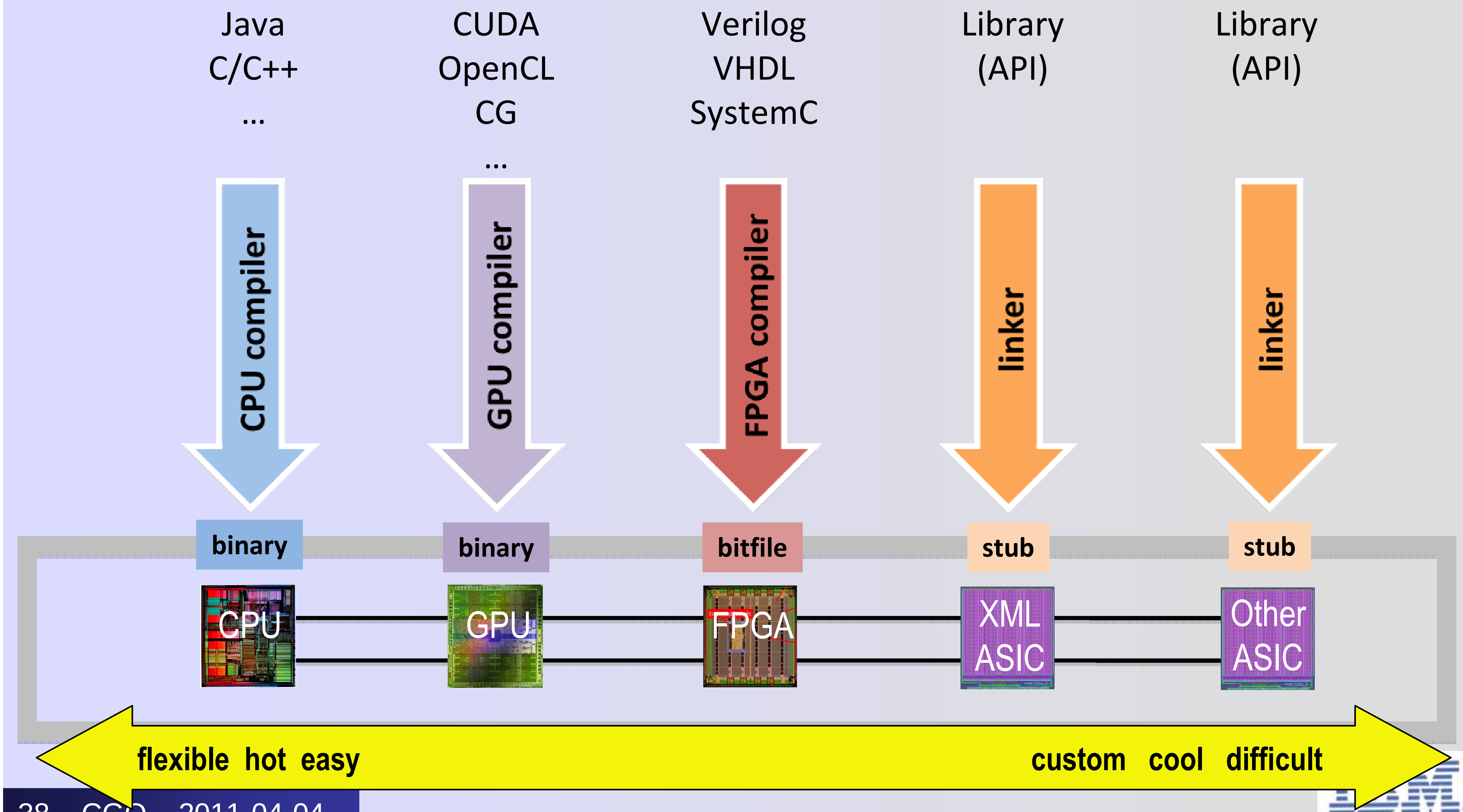
Problems

- Impractical growth of power and cooling
- Explosion of diverse architectures with massive parallelism
- Absence of a uniform abstraction
- Large productivity gap

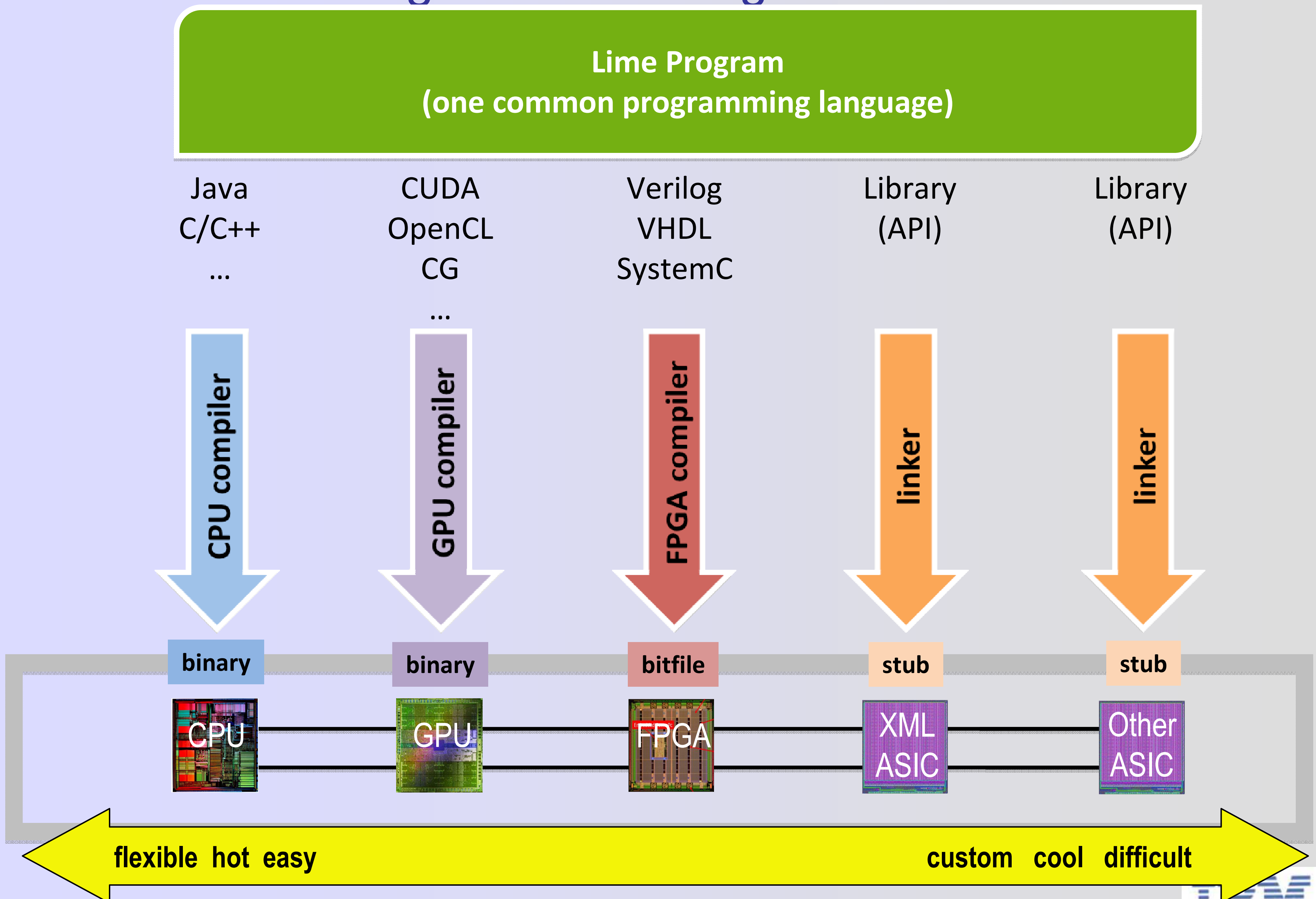
Liquid Metal Approach:

- **Lime:** A unified language for programming diverse architectures
- Run in a standard JVM, or compile to GPU and FPGA
- Automatically partition programs and execute each part where it runs best.
- Over time, make program placement more adaptive and dynamic
  - Until we can "JIT the hardware"
- Eclipse-based development environment
  - **Emphasis:** Programmer experience in the face of architectural diversity – *the new ISA?*
- Standard libraries analogous to Java Development Kit
- **Demos:** http://www.research.ibm.com/liquidmetal

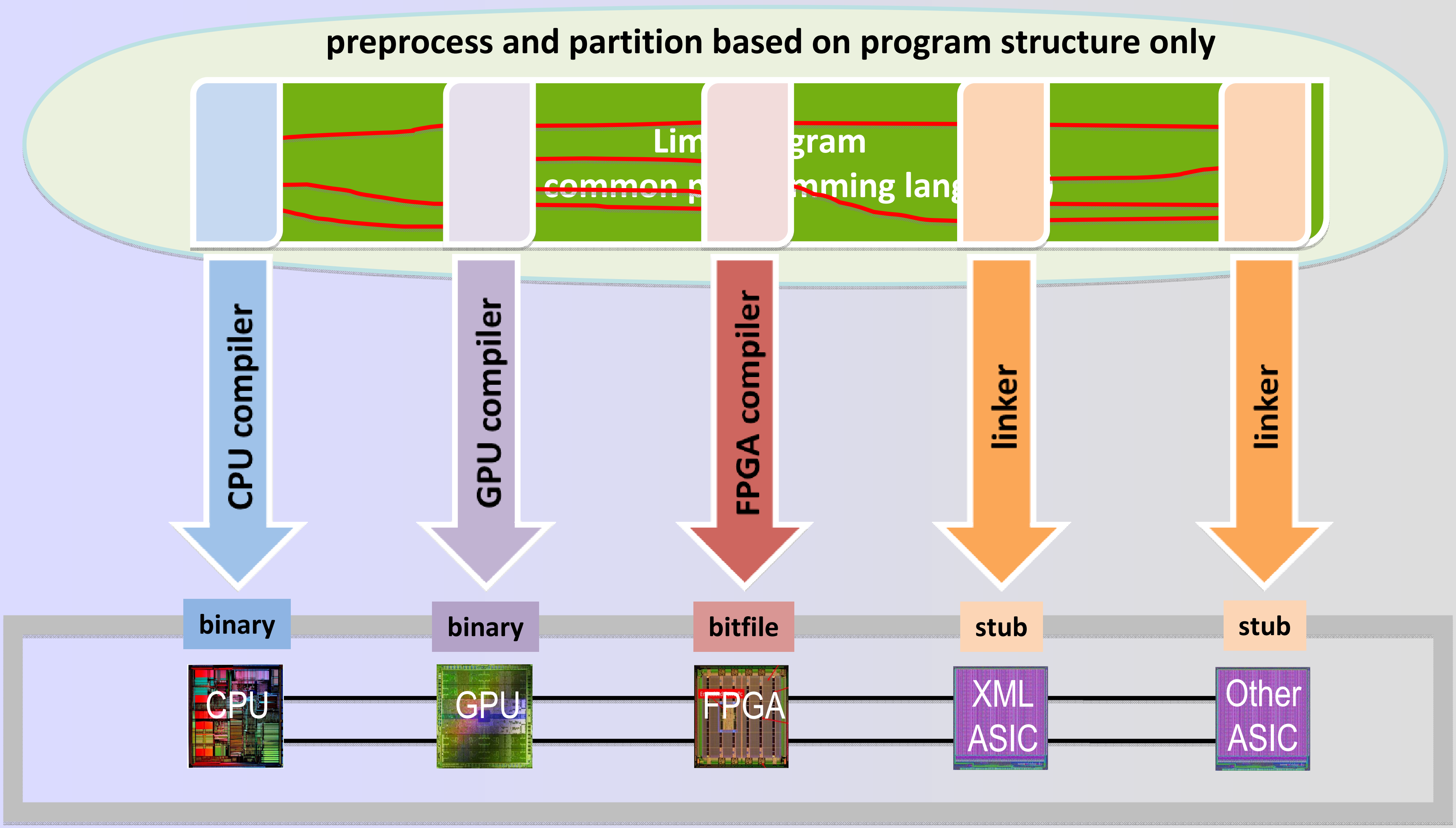
How do we Program a Heterogeneous Architecture?



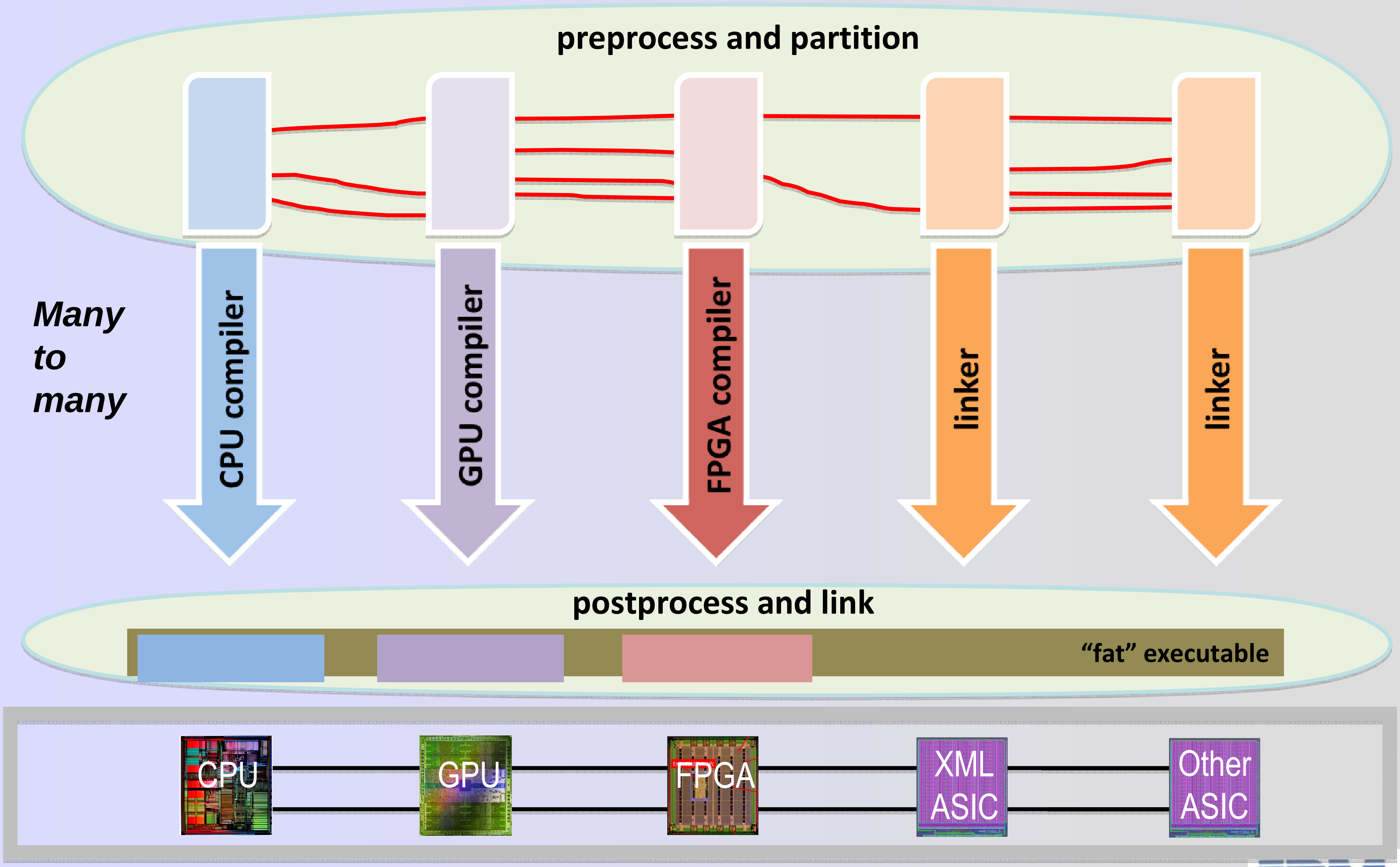
How do we Program a Heterogeneous Architecture?



Compiling Lime to Heterogeneous System



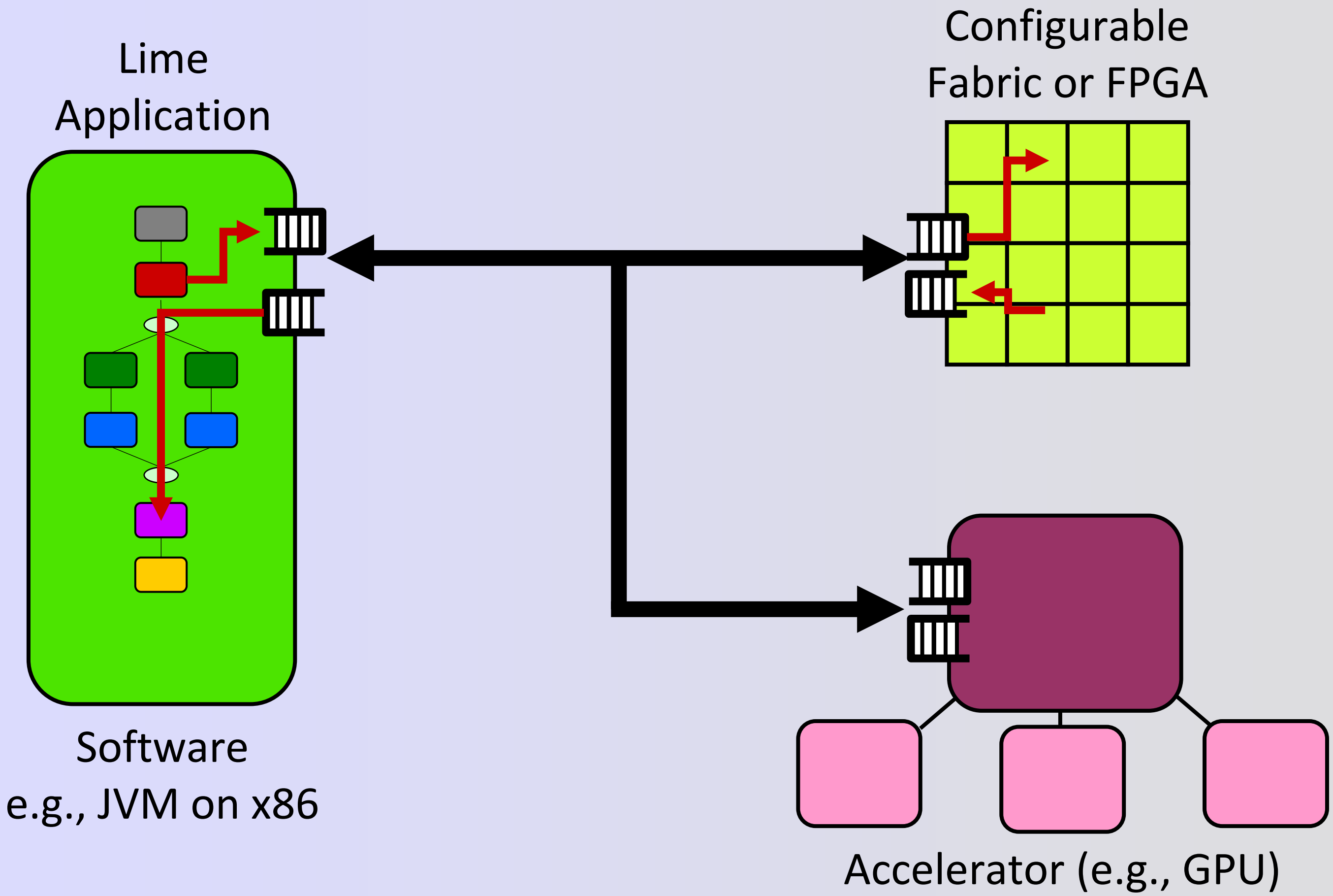
Compiling Lime to a Heterogeneous System



IBM

Dynamic Artifact Selection and Replacement

- Select among multiple (functionally equivalent) artifacts
  - Depending on runtime scenario and conditions



IBM

Queue Append

Verilog

```
always @(posedge clk or posedge reset) begin
      if (reset)
         con_free_tail   <= 6'd63 ;

      else if (p_state_r == terminate_con_state)
         con_free_tail   <= current_connection_ID_int ;
end

     …

end else if (n_state == terminate_con_state) begin
       free_ll_mem_en_A          <= 1'b1 ;
       free_ll_mem_BE_A          <= 2'b01 ;
       free_ll_mem_adr_A         <= con_free_tail ;
       free_ll_mem_wr_data_A     <= {8'h00, 2'b00,  current_connection_ID_int} ;

       free_ll_mem_en_B          <= 1'b1 ;
       free_ll_mem_BE_B          <= 2'b11 ;
       free_ll_mem_adr_B         <= current_connection_ID_int ;
       free_ll_mem_wr_data_B     <= {2'b00, con_free_tail, 2'b00, current_connection_ID_int} ;
```

Lime

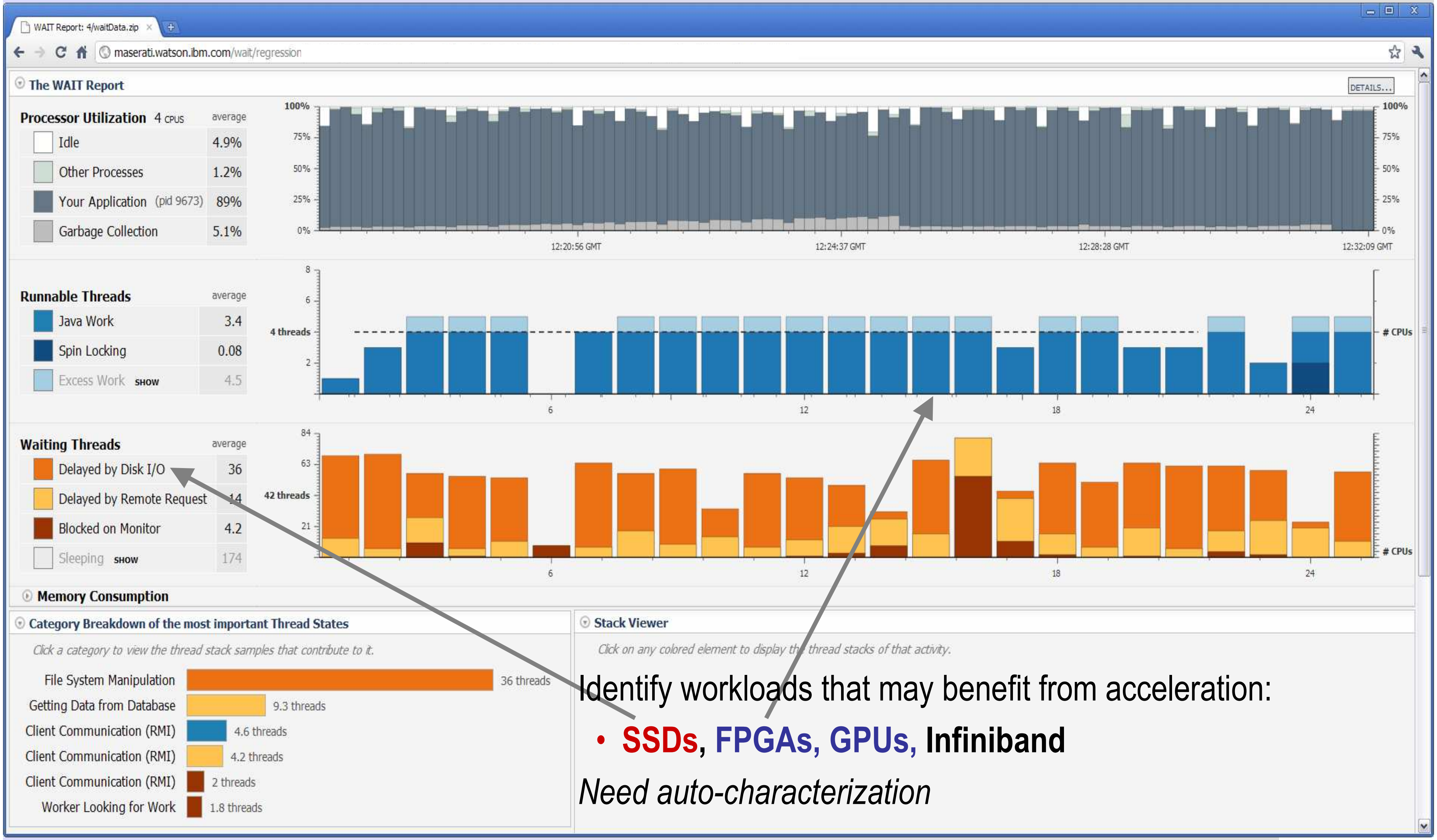
```
public local void addLast(E e) {
   if (empty) {
      head = tail = next[e] = prev[e] = e;
      empty = false;
   } else {
      next[tail] = e;
      prev[e] = tail;
      tail = e;
   }
}
```

Liquid Metal Perspective

- Current situation reminiscent of CISC vs RISC
  - Hardware primitives too complex for compiler to target from high level language
    - → Low-level languages like VHDL, Verilog, CUDA
    - **Less productive:** More lines of code for same function
- Could have library blocks of “RISC” from which efficient compilation performed.
  - **Problem:** Software variations and fine grain interactions
    - Blocks don’t do the function I want
    - Can’t compose blocks to efficiently perform function I want
  - → Difficult for this approach to succeed on a broad scale
- Semantic gap is hard to bridge
  - **Key:** Identify properties to help bridge the gap, e.g.
    - Streaming
    - Value types
    - Localness
    - Bounded arrays

- Lots of opportunities for CGO community. **Optimize:**
  - Loop transformations
  - Minimize hardware logic levels per FPGA clock cycle
  - Minimize communication between CPU, GPU, FPGA
  - Determine type of computing device best suited for each code fragment

Combining Liquid Metal and WAIT



Identify workloads that may benefit from acceleration:

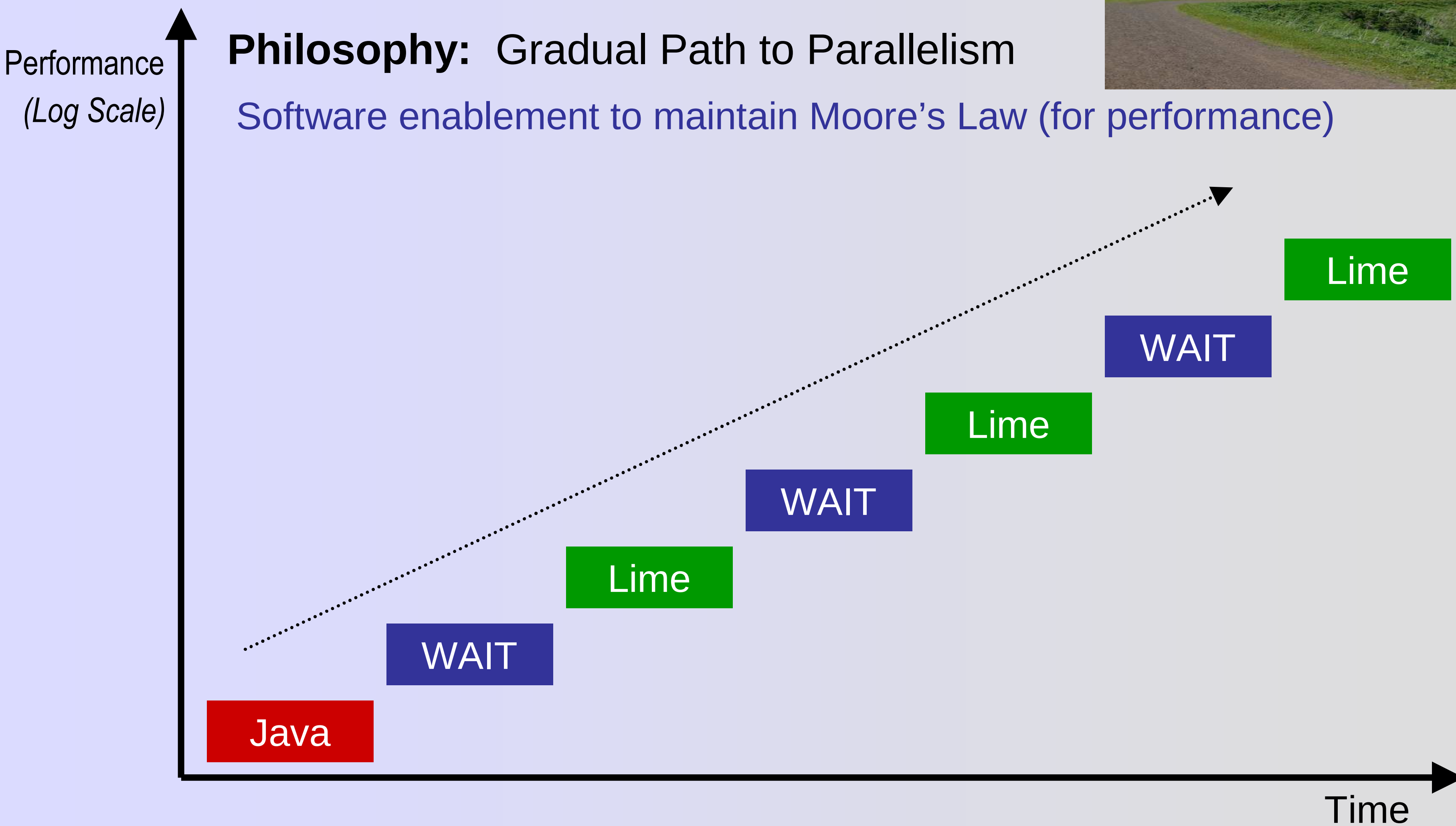
- **SSDs, FPGAs, GPUs, Infiniband**

Need auto-characterization

Incremental Refinements over Time

Philosophy: Gradual Path to Parallelism

Software enablement to maintain Moore's Law (for performance)



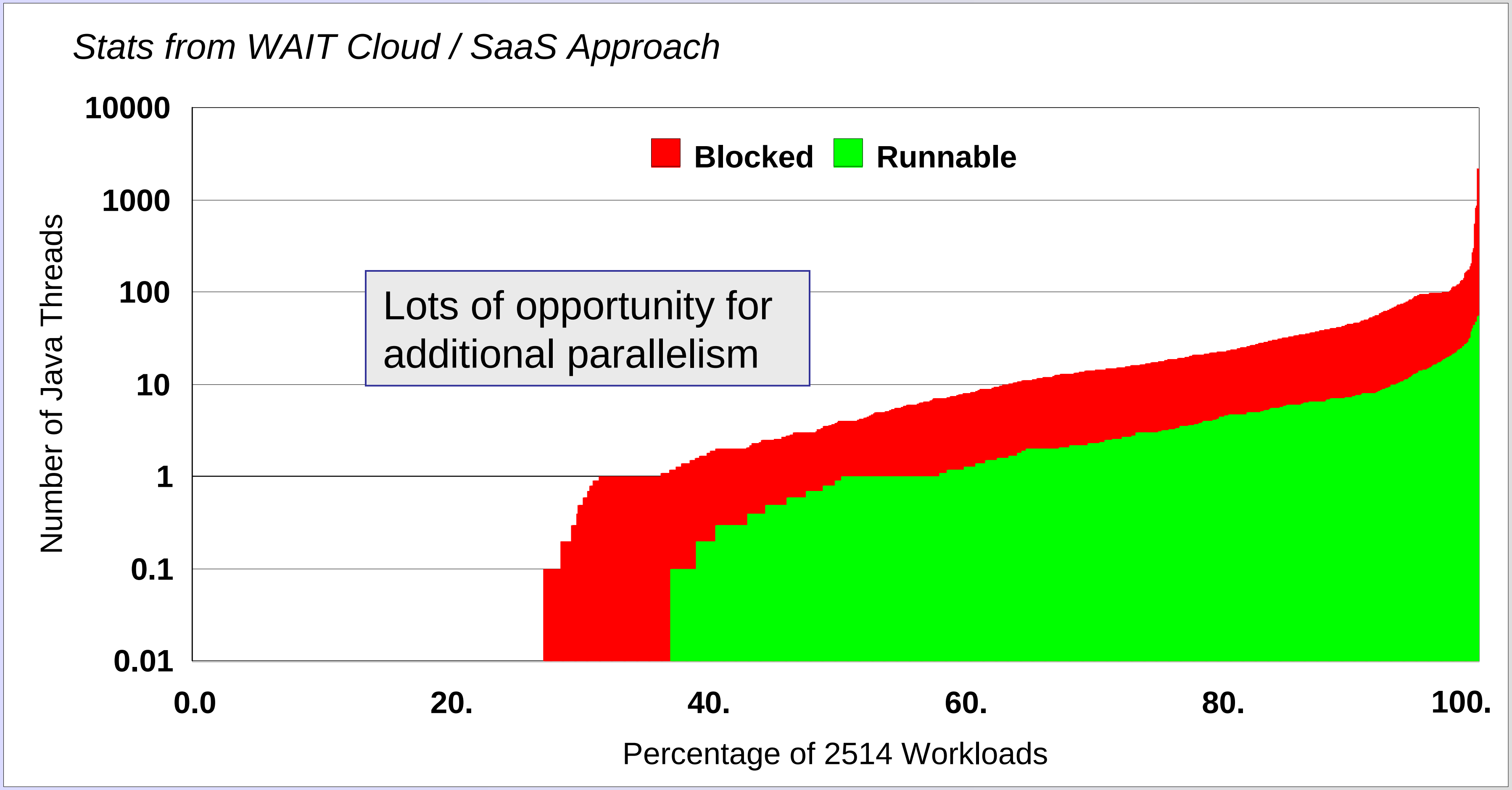
Making All of This Come to Fruition

- More uncertainty about future computing platforms than has been case during most of last 50 years.

➔

1. Important to be flexible.
2. Important to have access to lots of data.
   - In new era of efficiency and heterogeneity, systems are much less well understood.
   - Understanding and optimization will happen much faster with Cloud / **SaaS** (**S**oftware **a**s **a** **S**ervice)

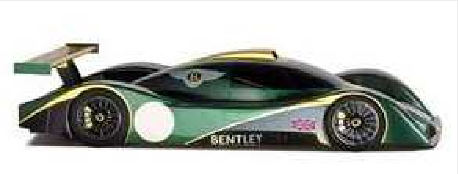
Thread Level Parallelism in Enterprise Workloads



Important to have access to lots of data.

Benefits to Users of Cloud Tools

- More efficient / Better performance
- Lower cost € $ ¥ £
- Faster performance improvement over time



- Easier management of complex systems
- Better customer service:

  – Agent can see customer problem.
  – Developers can quickly see problems hitting many customers.

Conclusion

- A gradual path to parallelism can be used for many technology generations.
  - Start with multi-threaded code under assumption of 2-way.
  - Tune (over time) as need more parallelism.
  - Cloud-based tooling.

Clean the mess

- Unless clock frequency starts improving, the need for new approaches is independent of Moore's Law.
  - Need to take advantage of increasing amounts of stuff.
  - Need to take advantage of increasingly heterogeneous stuff.
    - Cellphones to Servers
  - Need a new ISA.

Fix the mess

- **Optimize:** Lots of opportunities for CGO community:
  - Loop transformations
  - Minimize hardware logic levels per FPGA clock cycle
  - Minimize communication between CPU, GPU, FPGA
  - Determine type of computing device best suited for each code fragment
  - Automate the manual optimizations done using WAIT data

The End