Multi Core Playground

How can we get the most out of our modern CPU's?

Arden Thomas
Cincom's Smalltalk Product Manager

Cincom

Multi-Core Computers

- Are becoming ubiquitous….
- Quad cores from Intel & AMD are becoming commonplace
- 8,16,64 cores around the corner?

Cincom

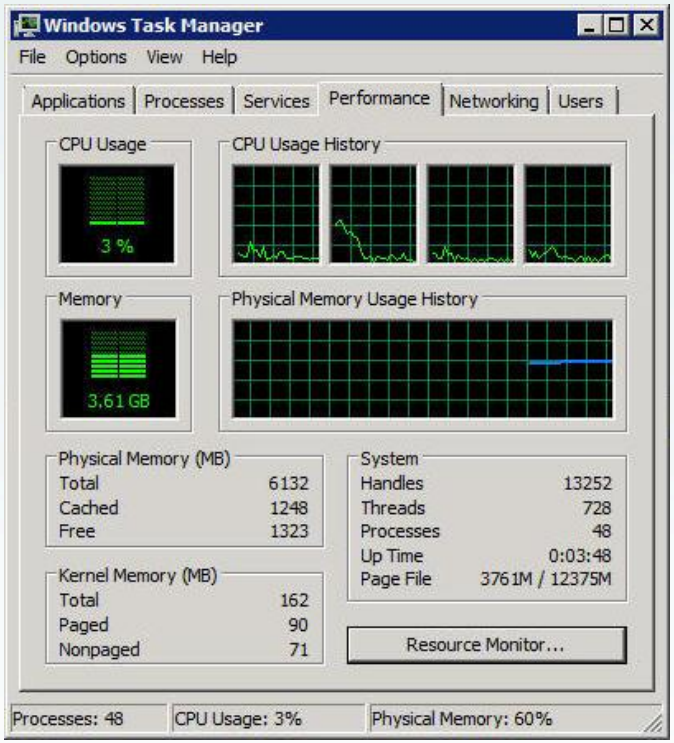
Cincom Smalltalk™ Roadmap Item

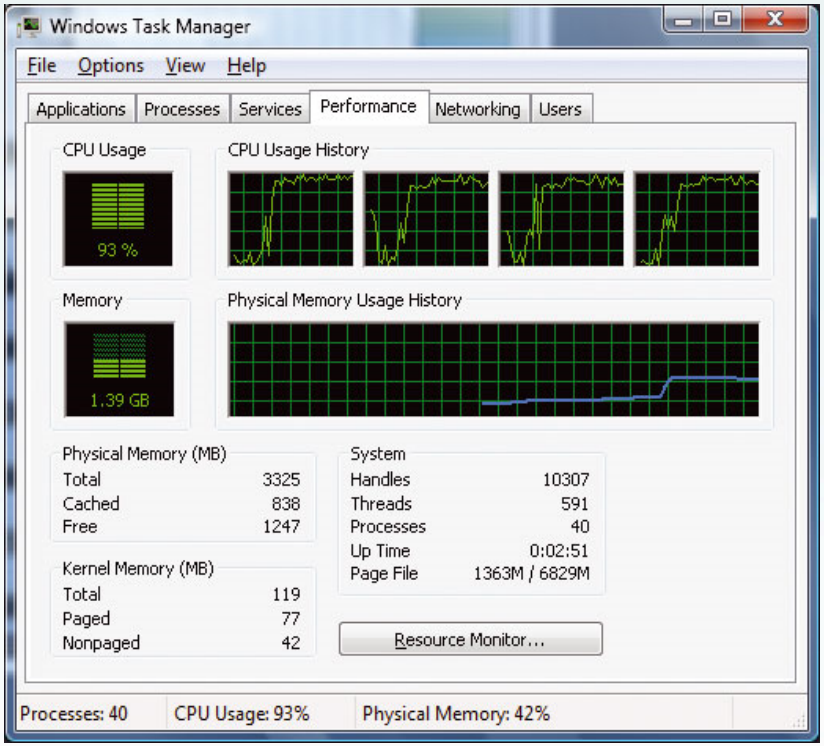
- *“Research ways to leverage Multi-Core computing”*
- This item was a magnet – lots of interest

Cincom

What is the Attraction?

- Making the most of what you have:





Cincom

Rear Admiral Grace Murray-Hopper

- Distinguished computer scientist
- Was there when they took a moth out of the relays of an early computer (*"getting the bugs out"*)
- Had a great ability to convey ideas in easy to grasp perspectives

Cincom

Grace Murray-Hopper

“When the farmer, using a horse, could not pull out a stump, he didn’t go back to the barn for a bigger horse,

he went back for another horse.”

Cincom

Using Multi-Core Computers

- “Team” of horses
- Type of concurrent programming
- On the same machine / OS
  - Generally faster
  - More options (i.e. shared memory, etc)

Cincom

…A ‘Small’ Matter of Programming….

- Most Concurrency is NOT EASY
- Concurrency problems and solutions have been studied for decades

Cincom

…A ‘Small’ Matter of Programming….

- Both AMD & Intel have donated money and personnel to Universities doing concurrency research
  - Specifically with the intent of increasing market demand for their products

Cincom

…A ‘Small’ Matter of Programming….

“As the Power of using
concurrency
increases linearly,
the complexity
increases exponentially”

Cincom

Approaches to Using Multi-Cores

Multiple Applications

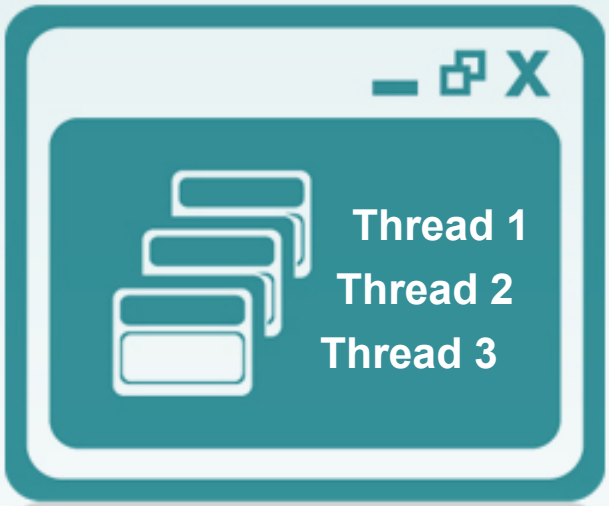
Cincom

Approaches to Using Multi-Cores

- Multiple Applications
- Advantages
  - Simple
  - Easy
  - Effective
  - Minimal contention
- Disadvantages
  - Multiple independent applications needed
  - Not necessarily using multi-cores for one problem

Cincom

Approaches to Using Multi-Cores



Multiple Process Threads in a Single Application

Core 1 | Core 2 | Core 3 | Core 4

Cincom

Approaches to Using Multi-Cores

- Multiple Process threads in a single application
- Advantages
  - Threads have access to the same object space
  - Can be effective concurrency
- Disadvantages
  - Object contention and overhead
  - Usually significant added complexity
  - Threads (if native) can take down the whole application in severe error situations

Cincom

Approaches to Using Multi-Cores

- Multiple Process threads in a single application
  - Multiple “green” (non native) threads available in VisualWorks/ObjectStudio8
- This can still be very effective
  - Modeling producer/consumer problems
  - Effective example later

Cincom

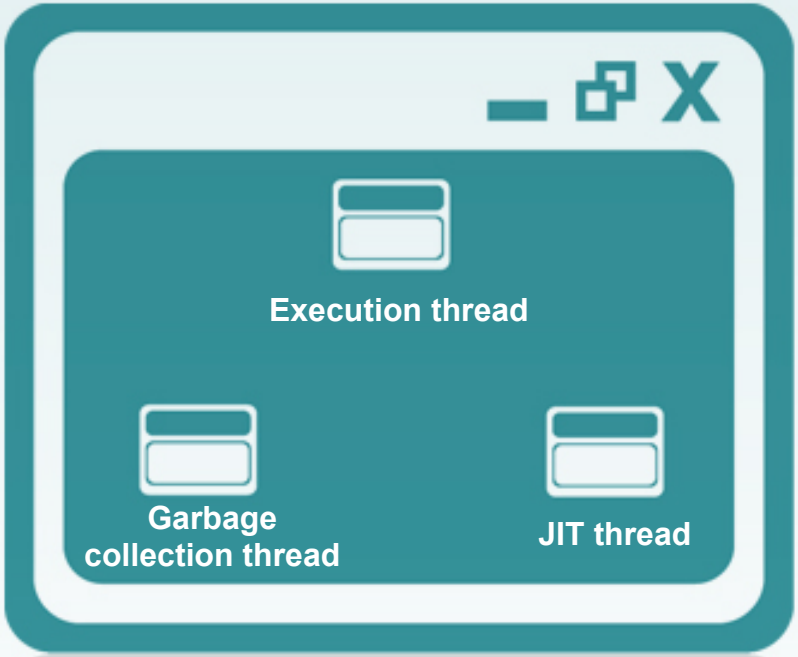
Some Smalltalk Objects for Concurrency

- Process
- Semaphore
- Promise (a kind of ‘Future’)
- SharedQueue

Cincom

Approaches to Using Multi-Cores

Multiple Process Threads in a CST VM



Core 1 Core 2 Core 3 Core 4

Cincom

Approaches to Using Multi-Cores

- Multiple Process threads in a CST VM

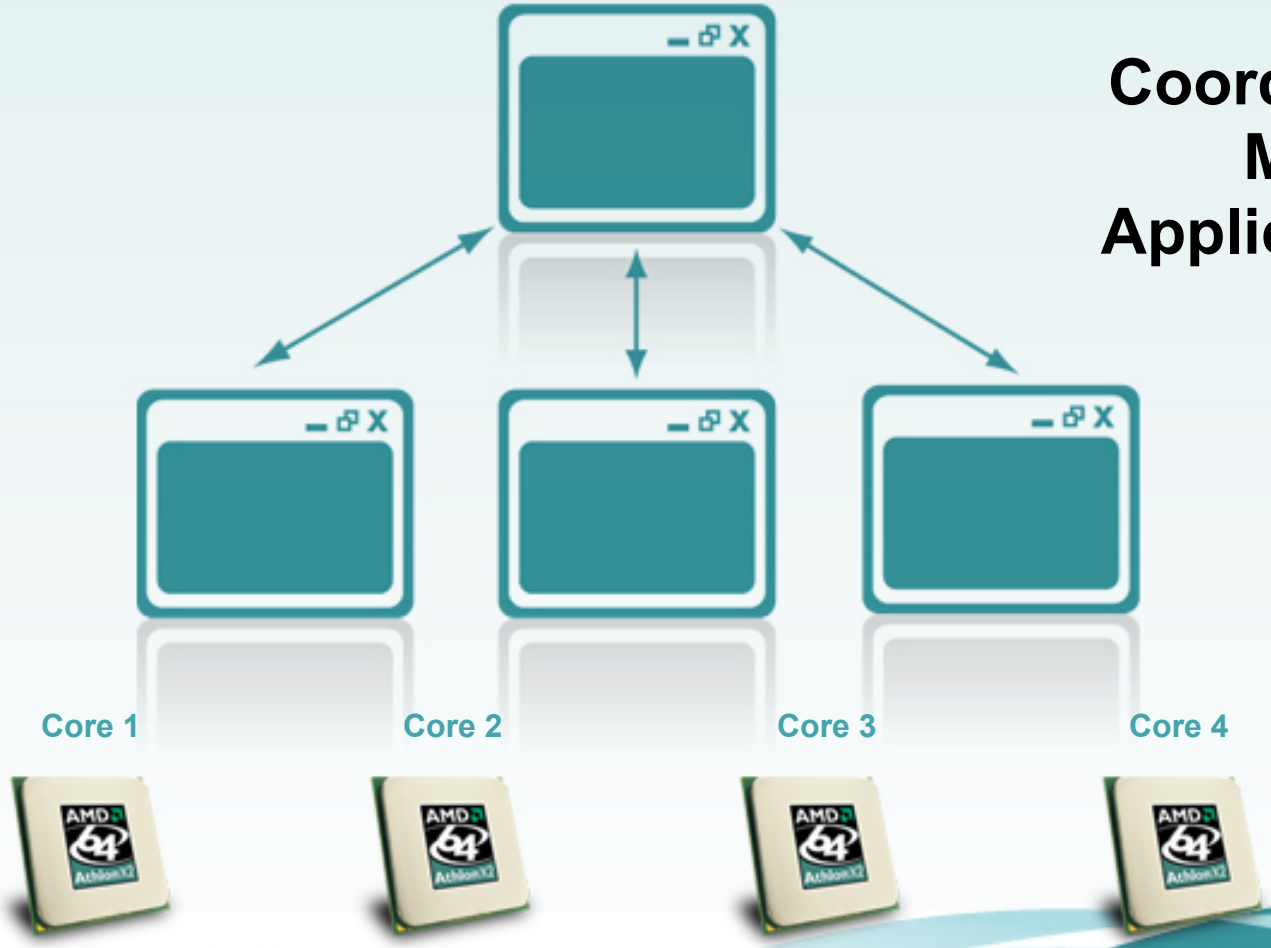
  Note that the idea is just an example for discussion, product management brainstorming

- Advantages
  - Use of multi-cores even with single threaded applications
- Disadvantages
  - Time & resources to develop VM
  - Feasibility and stability questions

Cincom

Approaches to Using Multi-Cores

Coordinated Multiple Applications



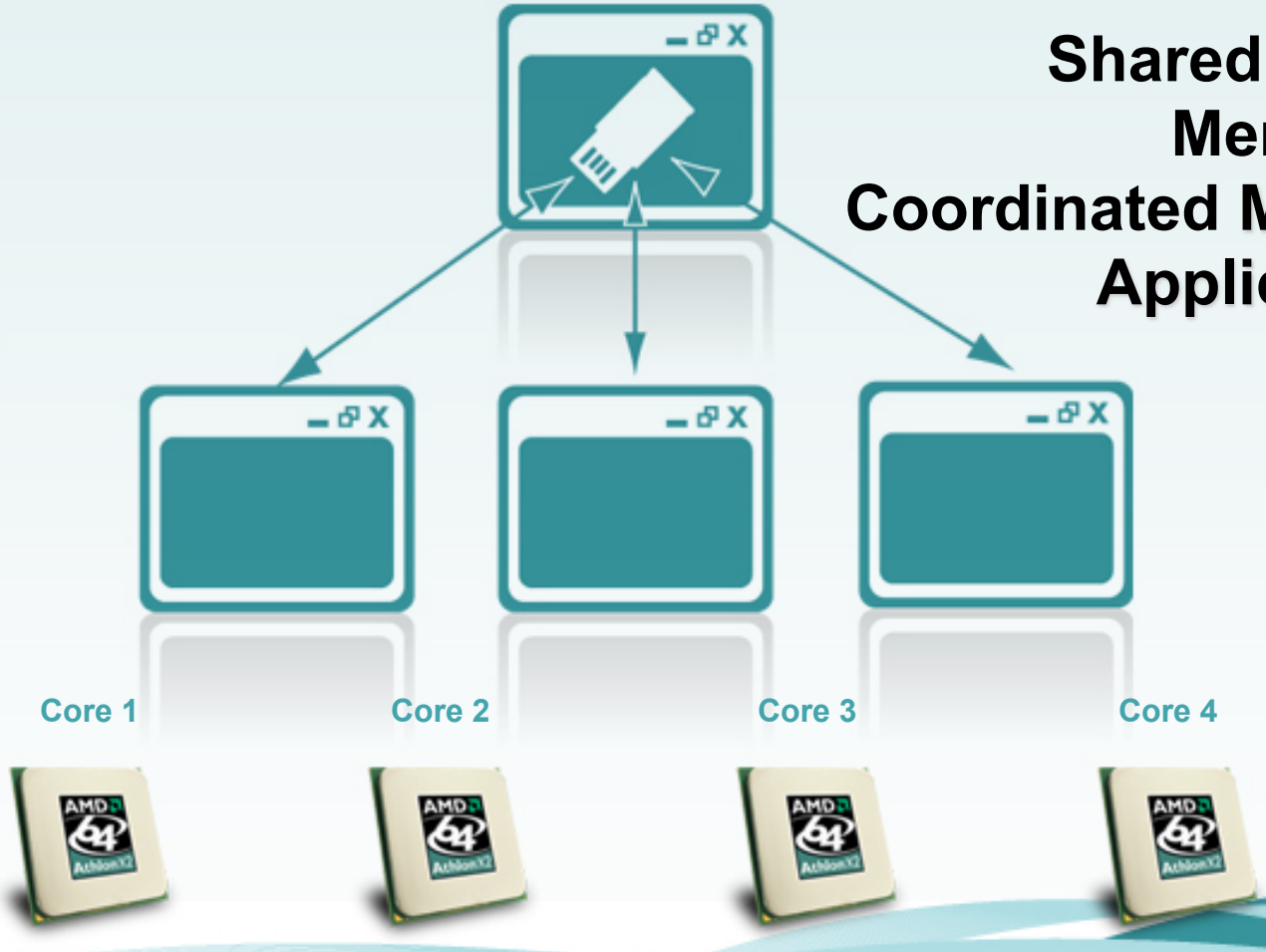
Cincom

Approaches to Using Multi-Cores

- Coordinated Multiple Applications
- Advantages
  - Smaller changes in code needed
  - Fairly Easy & Effective
  - Could be Scaled to number of Cores
  - Fewer contention issues
  - Doable without VM changes
- Disadvantages

Cincom

Approaches to Using Multi-Cores



Shared Object Memory & Coordinated Multiple Applications

Cincom

Approaches to Using Multi-Cores

- Shared Object Memory with Coordinated Multiple Applications
- Advantages
  - Can solve a broader set of problems
  - Can bring multiple cores to bear on the same object set
- Disadvantages
  - Garbage collection complications
  - Need to coordinate or manage sharing (traditional concurrency problems)

Cincom

Product Management Requirements

- Research into ways to utilize multi-core computers
  - We can add multiple facilities & abilities over time
- Something "Smalltalk simple“
  - A simple mechanism that is simple, effective, and high level
  - Not Smalltalk implementation of generally hard things
- Avoids most of the traditional problems and difficulties if possible
  - Minimize contention, locking, ability to deadlock
- Is flexible enough to be the basis for more sophisticated solutions as situations warrant

Cincom

Polycephaly

- Engineering experiment
- Assists in starting and managing headless images
  - Handing them work
  - Transporting the results

Cincom

Polycephaly – What it Isn't

We have not magically invented some panacea to the difficult issue of concurrency

Cincom

Polycephaly – What it Is

- A simple framework that allows a subset of concurrency problems to be solved
- For this class of problems, it works rather nicely

Cincom

Experiment

- Take some code that performs a task
- See how overall time to perform the task could be improved, using concurrency
- Use the Polycephaly framework
- Goal:
  - See if substantial improvements, via concurrency could be made with a small, minor and simple amount of effort.

Cincom

Task

- Loading market security information
  - Stocks
  - Mutual Funds
  - ETFs
- Two Sources
  - Files from Nasdaq
    - For NYSE, AMEX, Nasdaq
  - HTTP
    - Mutual Funds (around 24,000!)

Cincom

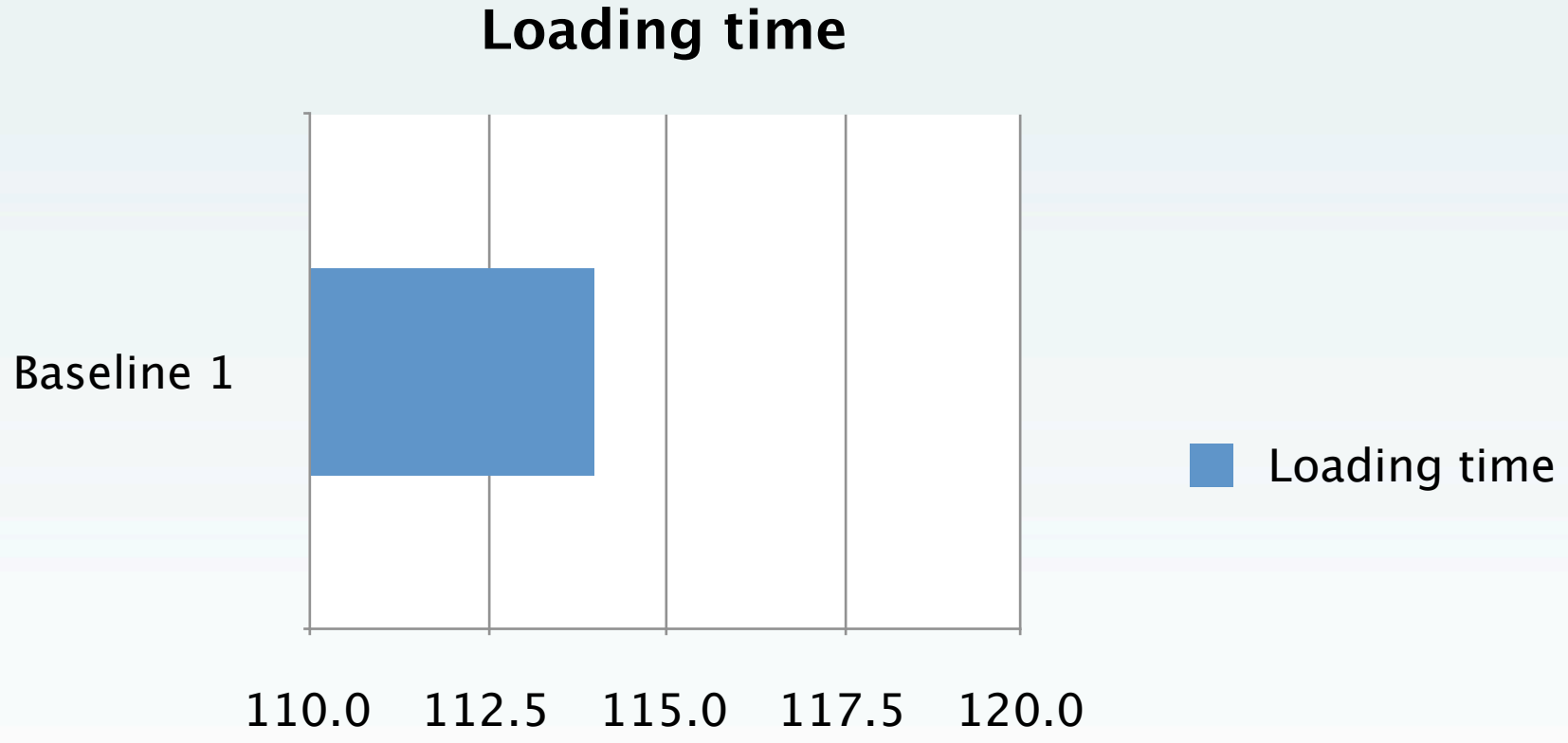
Baseline Code

- Original sequential load code:

```
nyse := Security loadNYSE.
amex := Security loadAMEX.
nasd := Security loadNasdaq.
funds := MutualFund load.
etfs := ETF load.
```

Cincom

Time Baseline - Code Run Sequentially



Cincom

Experiment I

- Make the five loads work concurrently using Polycephaly

Cincom

Experiment I

- Concurrent load code:
  nyseLoad := self promise: ‘Security loadNYSE'.
  amexLoad := self promise: ‘Security loadAMEX'.
  nasdLoad := self promise: ‘Security loadNasdaq'.
  fundsLoad := self promise: ‘MutualFund load'.
  etfsLoad := self promise: ‘ETF load'.

- nyse := nyseLoad value.
  amex := amexLoad value.
  nasd := nasdLoad value.

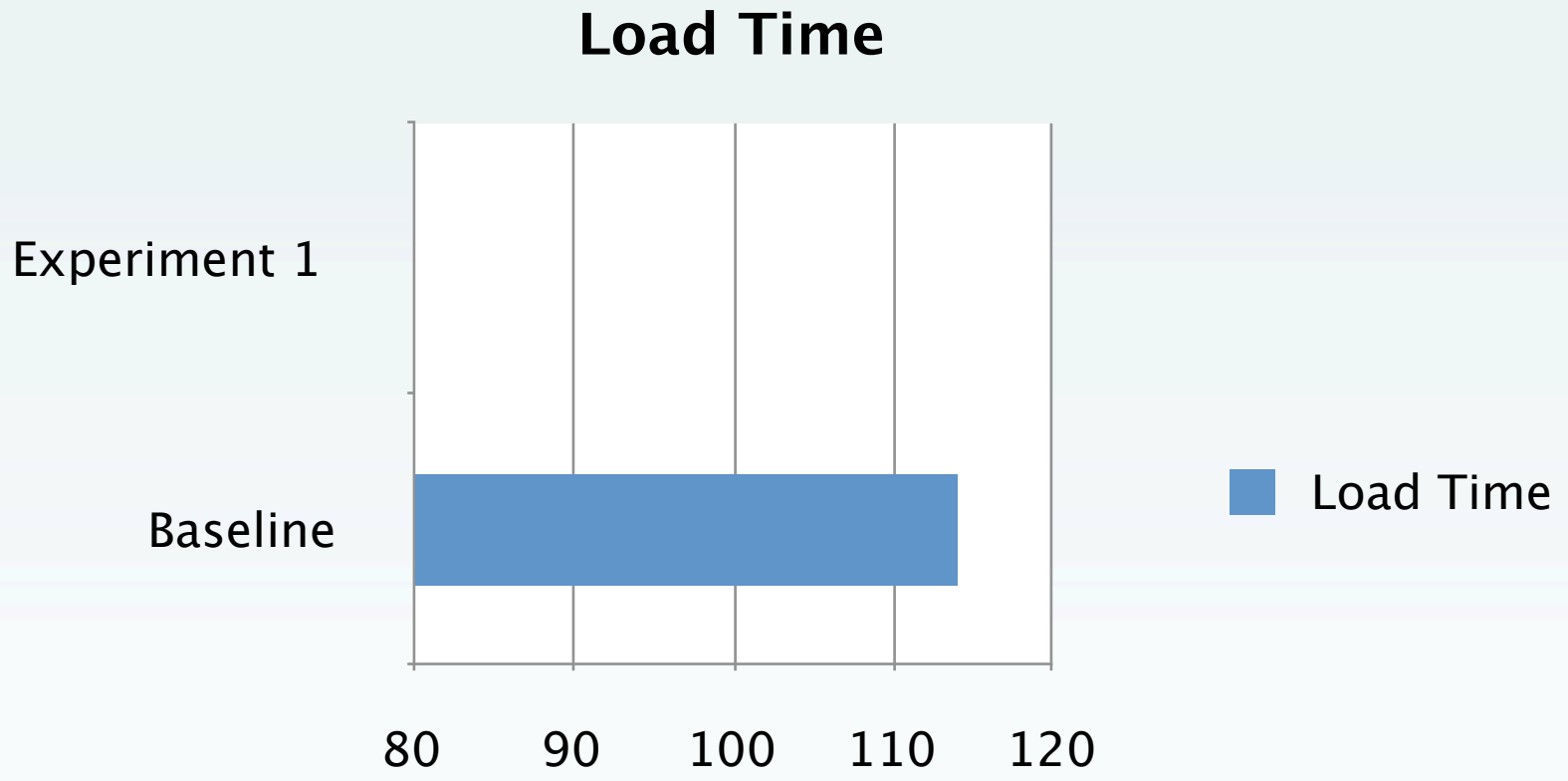
Cincom

Experiment I

- promise: aString
  | machine |
  machine := VirtualMachine new.
  ^[| val |
  val := machine doit: aString.
  machine release.
  val] promise

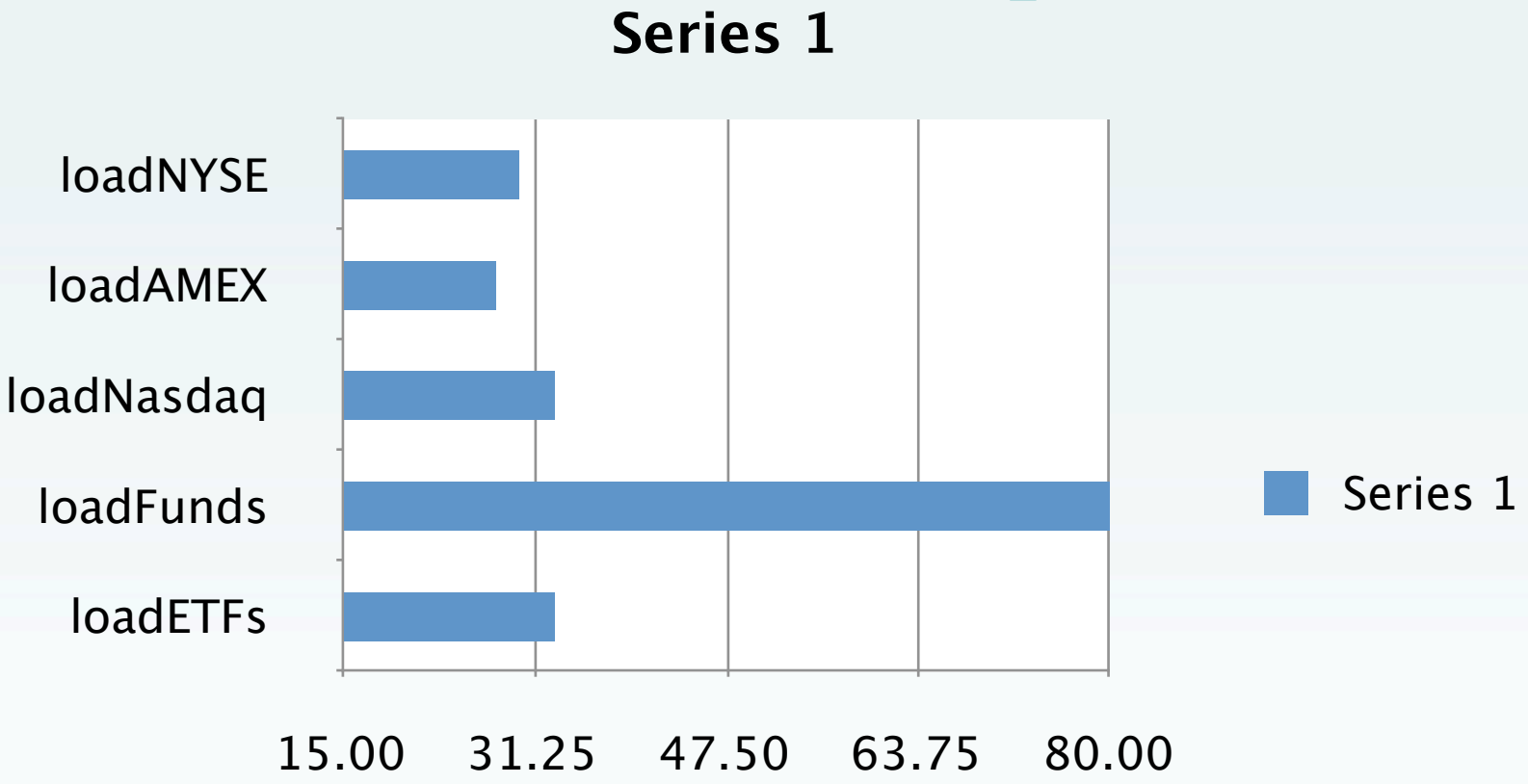
Cincom

Experiment I



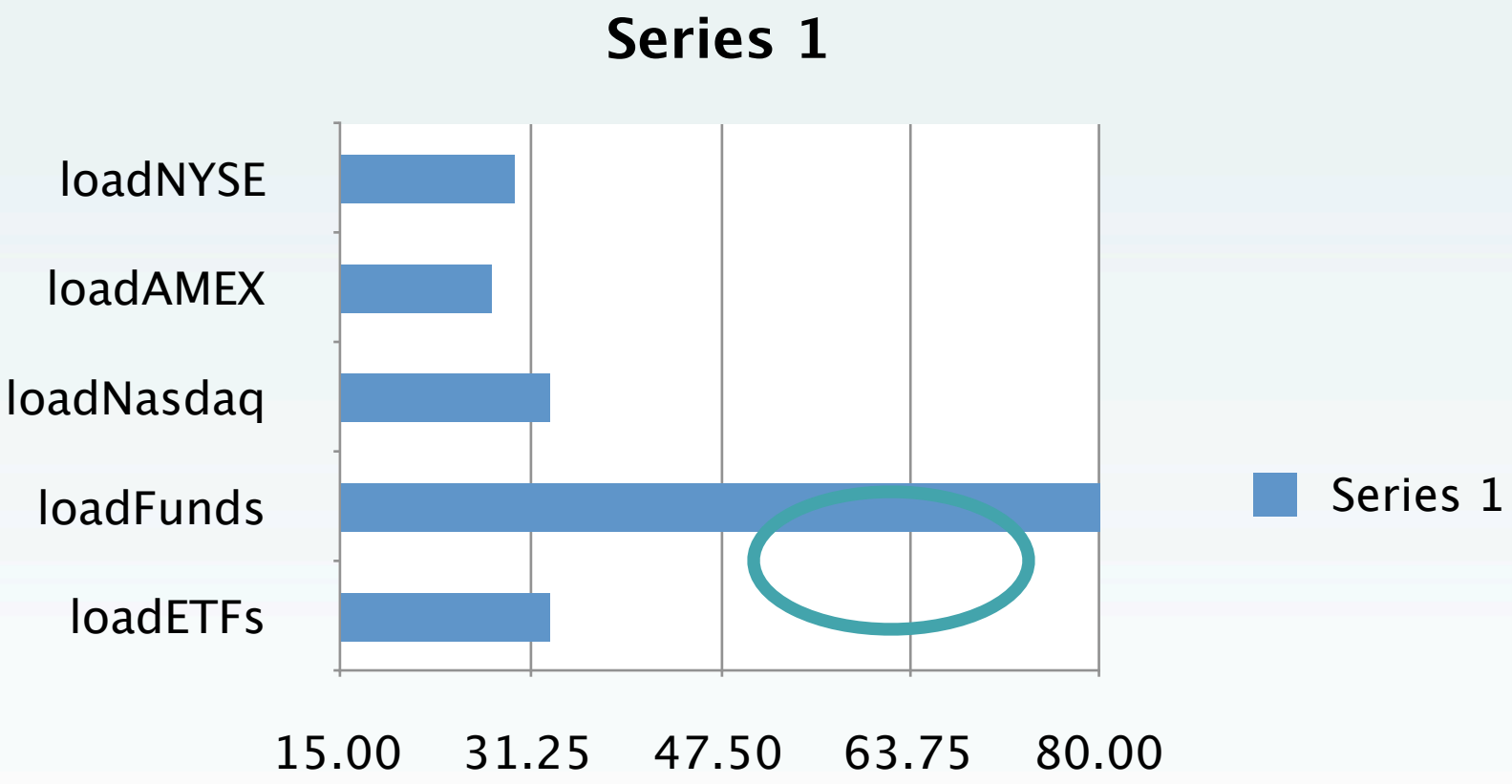
Cincom

Experiment I



Cincom

Experiment I - Bottleneck



Cincom

Experiment II – Addressing the Bottleneck

- loadFunds took the majority of the time
- Loads information for over 24,000 Mutual Funds
- Loads via http
- Loads all Funds starting with ‘A’, then ‘B’ ….
- So try making each letter an independent task ….

Cincom

Experiment II – Addressing the Bottleneck

- Create n “drones” (start n images)
- Each drones grabs a letter and processes it
  - Then returns the results to the requesting image
- When done it asks for another letter
- For n from 3 to 20

Cincom

Experiment II – Addressing the Bottleneck

- Time with: 3 drones: 30244
  Time with: 4 drones: 22373
  Time with: 5 drones: 20674
  Time with: 6 drones: 18271
  Time with: 7 drones: 18570
  Time with: 8 drones: 17619
  Time with: 9 drones: 17802
  Time with: 10 drones: 17691
  Time with: 11 drones: 19558
- Time with: 12 drones: 18905
  Time with: 13 drones: 17658
  Time with: 14 drones: 19696
  Time with: 15 drones: 21028
  Time with: 16 drones: 19704
  Time with: 17 drones: 21899
  Time with: 18 drones: 19400
  Time with: 19 drones: 19884
  Time with: 20 drones: 20698

Cincom

Experiment II – Addressing the Bottleneck

- Points to note about the solution
  - Times include
    - Start-up of drone VM's
    - Object transport time

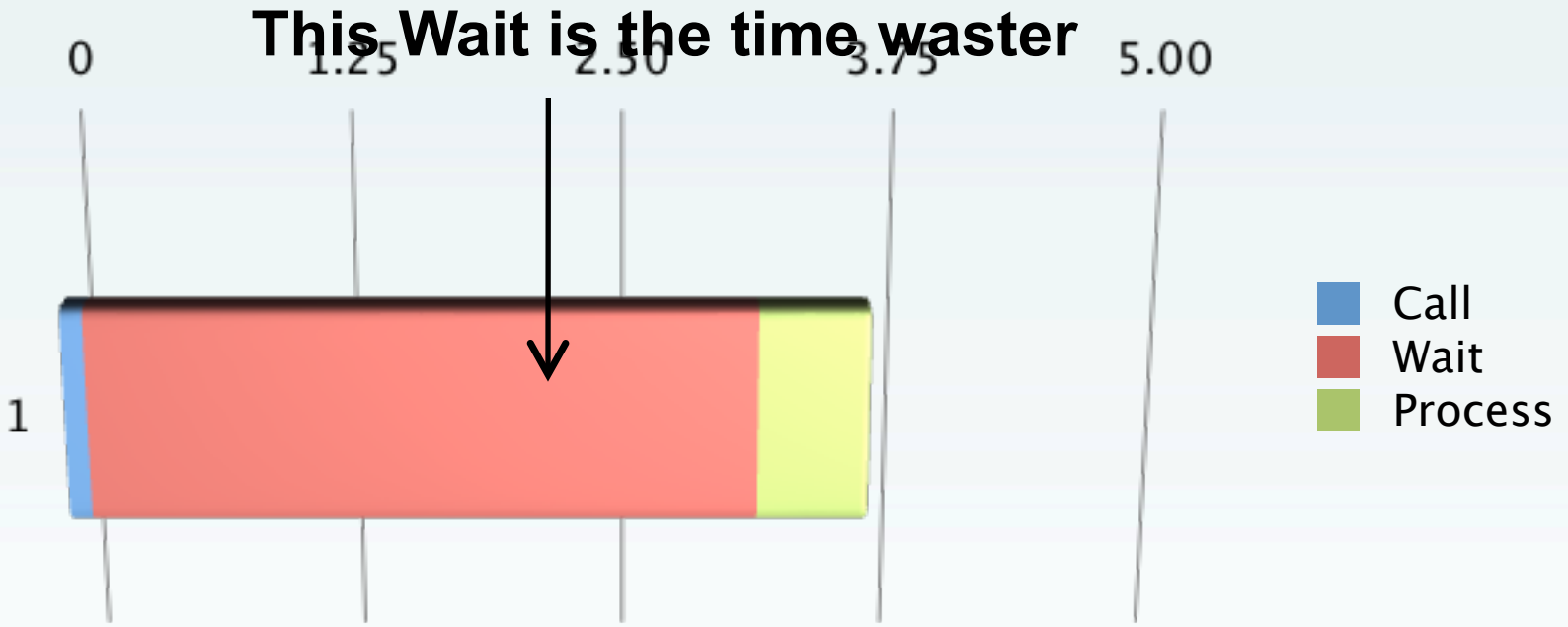
Cincom

Experiment III – Addressing the Bottleneck, Revisited

- Lets take another look at the problem
- 26 http calls
  - Call
  - Wait
  - Process
  - Repeat

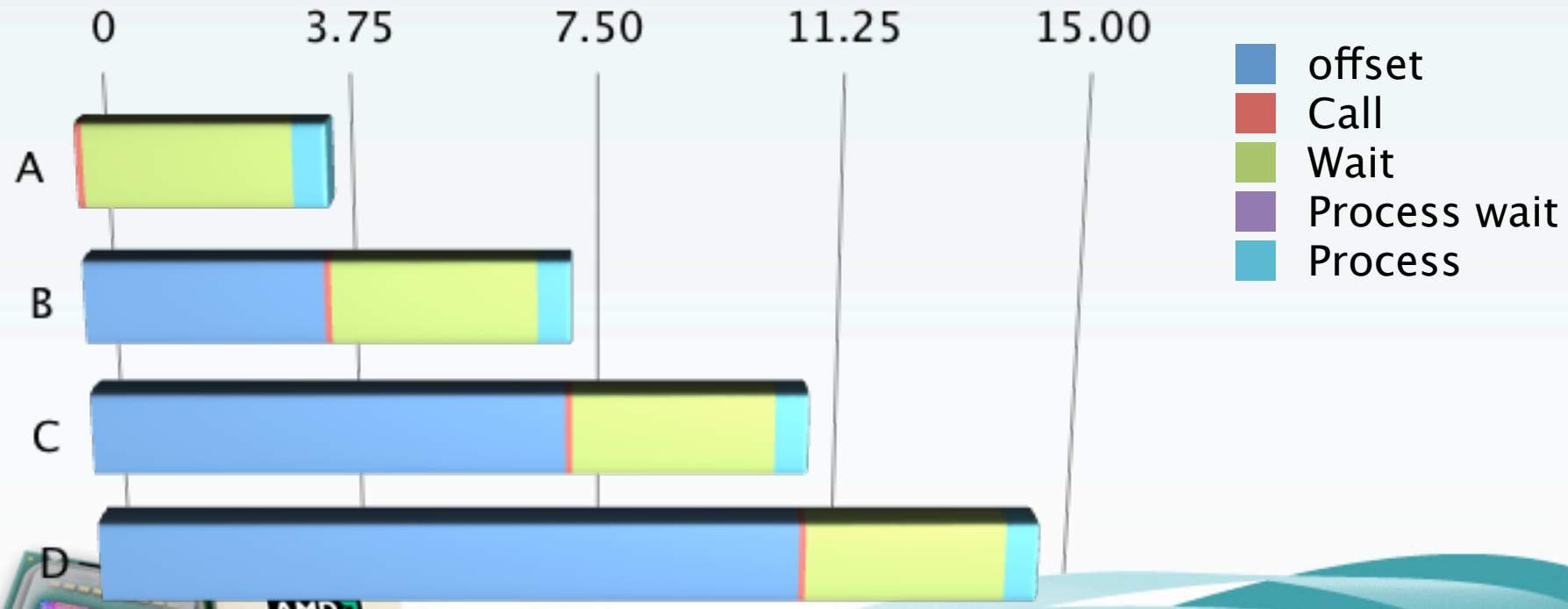
Cincom

Bottleneck Revisited– the Orange Issue



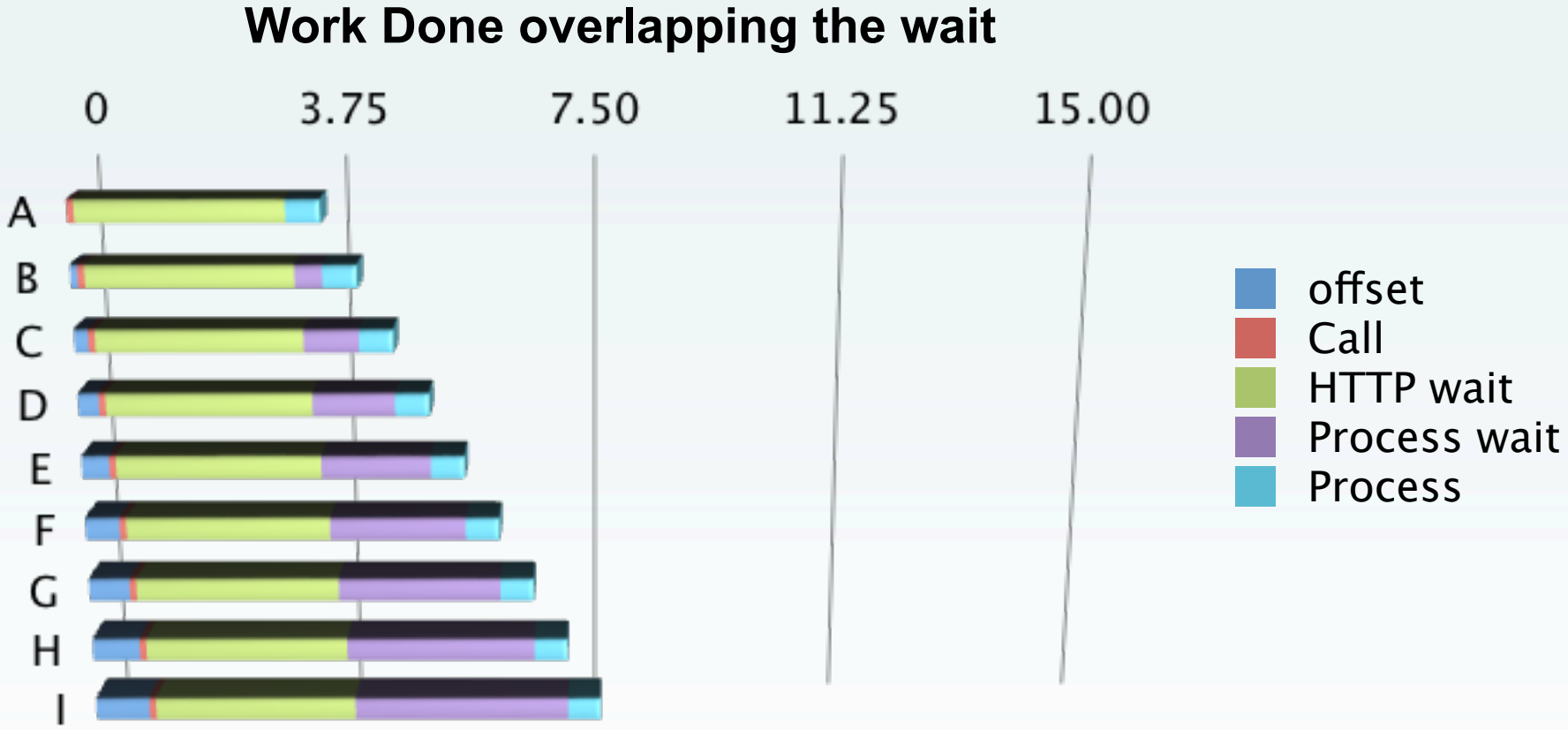
Cincom

Bottleneck Revisited- Sequence



Cincom

Bottleneck Revisited – Overlap Solution



Cincom

Code for the Curious ....

```
1 to: n do: [:i |
[ | char |
    [char := queue nextAvailable. char isNil]
       whileFalse: [results nextPut: (Alpha.MutualFund  loadForChar: char )].
    done signal] fork].
n timesRepeat: [done wait].
```

Cincom

Bottleneck Redux – Overlap Solution

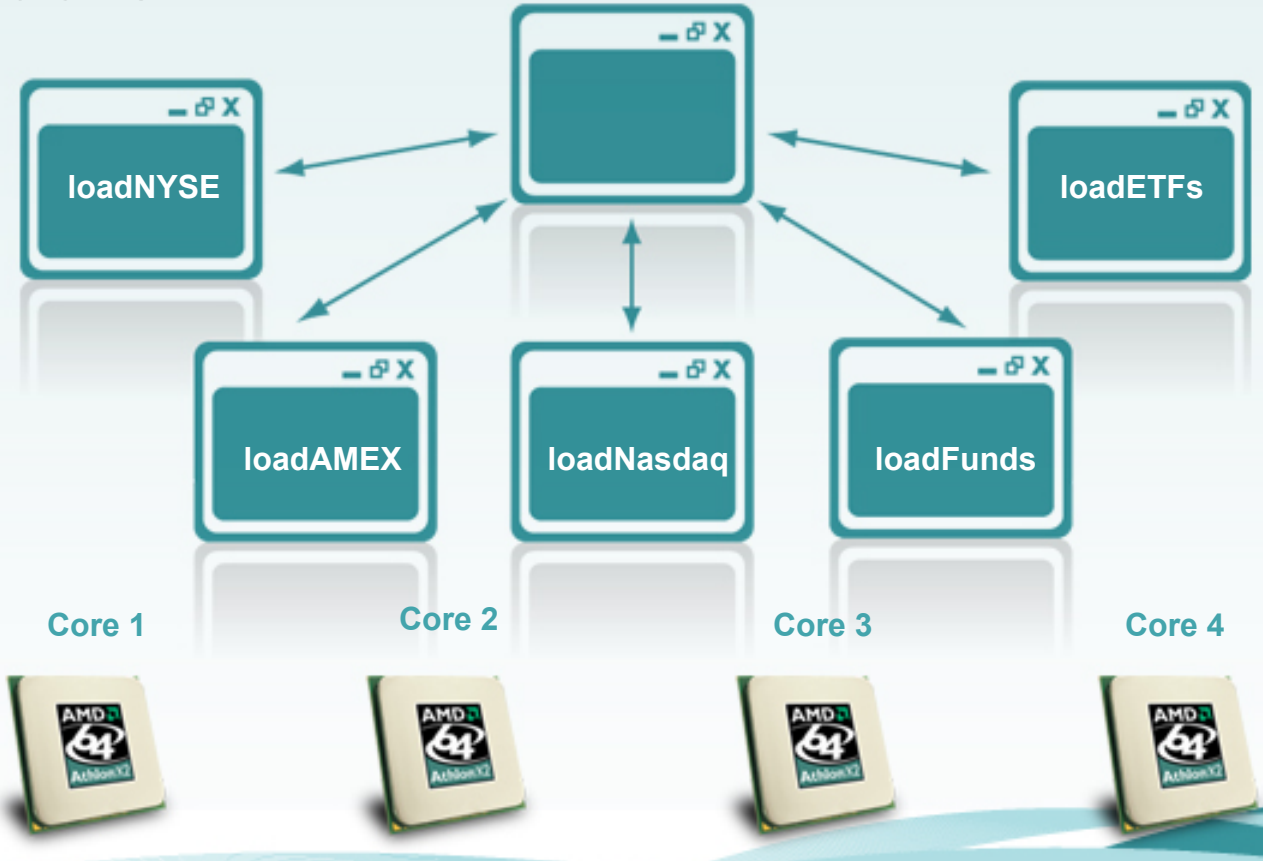
- Results
  - Used 13 green threads
  - Reduced the load time to *15 seconds*!
  - All in one image

What about 26 threads?

Cincom

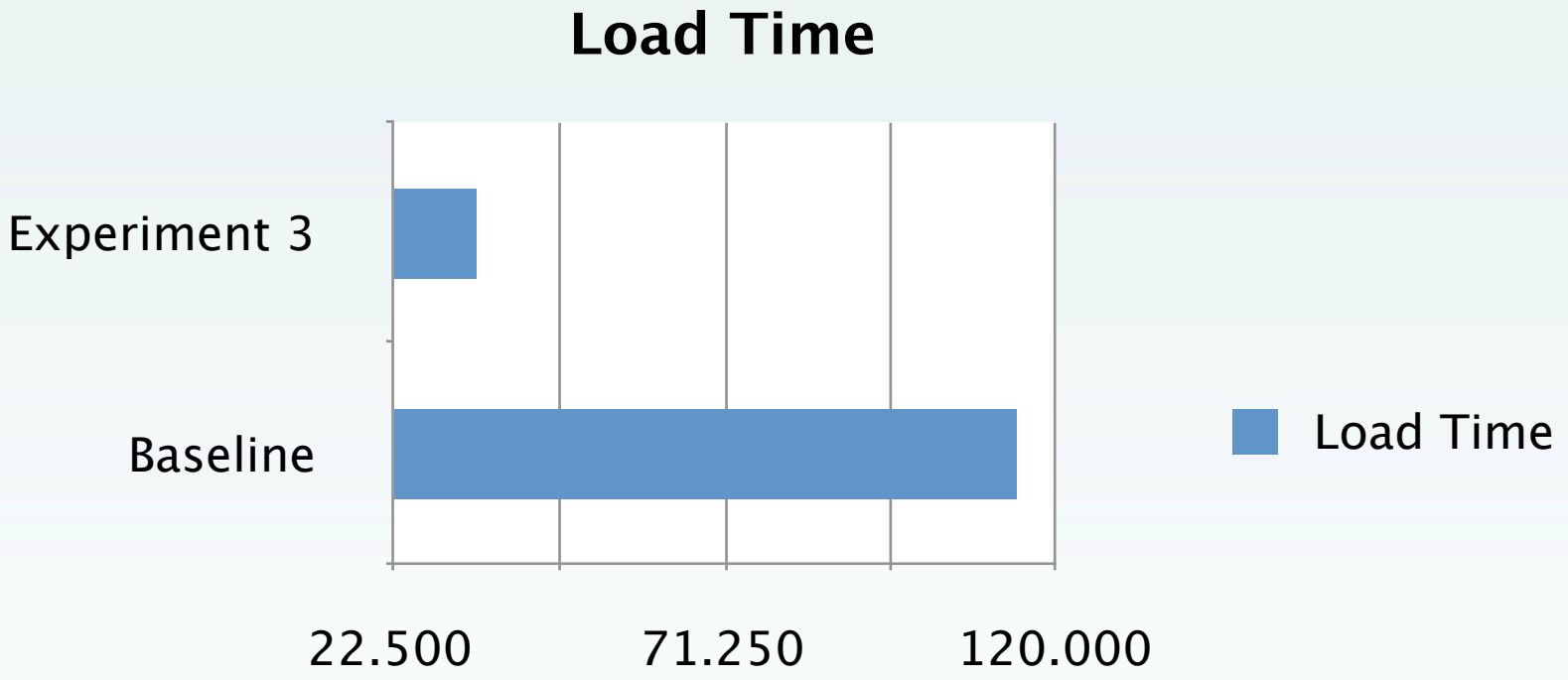
Putting it All Together

Coordinated Multiple Applications



Cincom

Putting it All Together



Cincom

Putting it All Together

- Final overall load time using 5 drones, along with threaded http calls for loadFund, saw overall load time improve from 114 seconds to 35 Seconds
  - The 35 seconds *includes* all image startup and transport time

Cincom

Some Observations

- Measure the elapsed time of work - it helps to know where the problems are in order to address them
- There are multiple way to do concurrency. In one experiment, I backed off using the new framework, because a simpler approach worked better. This was a pleasant surprise - the green thread model was actually very well suited for that problem. I ended up combining both for a superior solution.
- Strive for simplicity. Remember if it gets out of hand, concurrency can get ugly and difficult very quickly. Follow the KISS principle (Keep it Simple, Smalltalk ;-)).

Cincom

Some Observations

- The launch of drone images was faster than I expected
- Drone images are instances of the main image
- As mentioned in the commentary, my measurements included startup and shutdown of images, a (overly?) conservative approach that may be atypical to how many use this technology. Results would be even better had I not included that.

Cincom

Some Observations

- The experimental work has some pieces that I think show a lot of foresight. Mechanisms for handling some common issues. Like what?
  - If I had a dozen drone images running and lost my references and the ability to close them, closing the main image shuts everything down. Things like that save a lot of time, avoid aggravation, and make it fun.
  - If I started a local process, which in turn started a drone, then later I terminated that process, the drone would be terminated.
  - Remote errors can be retrieved locally

Cincom

Conclusions

- I believe my requirements of providing one means of concurrency with maximum gain and minimum pain are met
- My expectations on the simplicity and robustness of the experimental framework were surpassed

Cincom