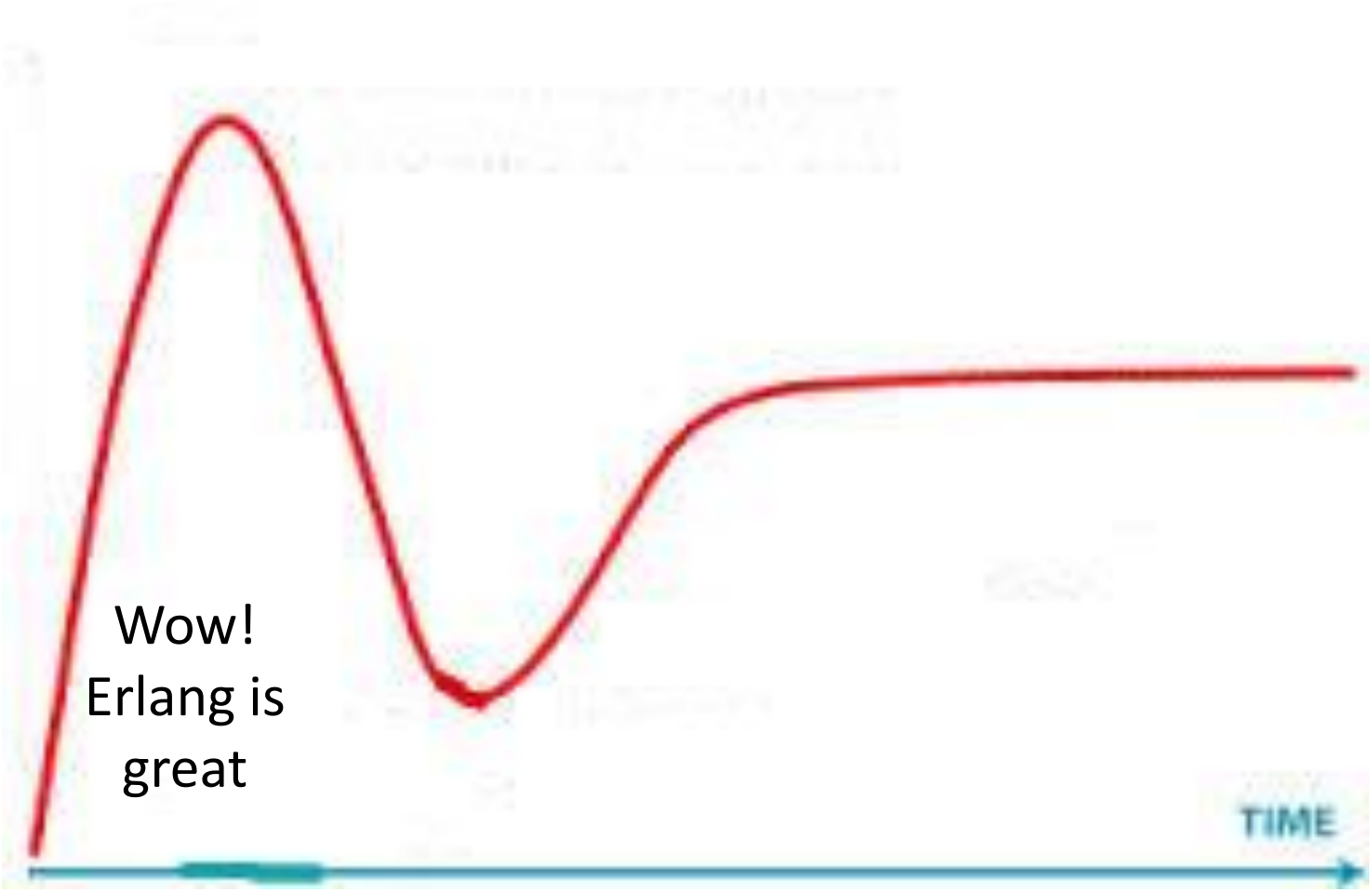
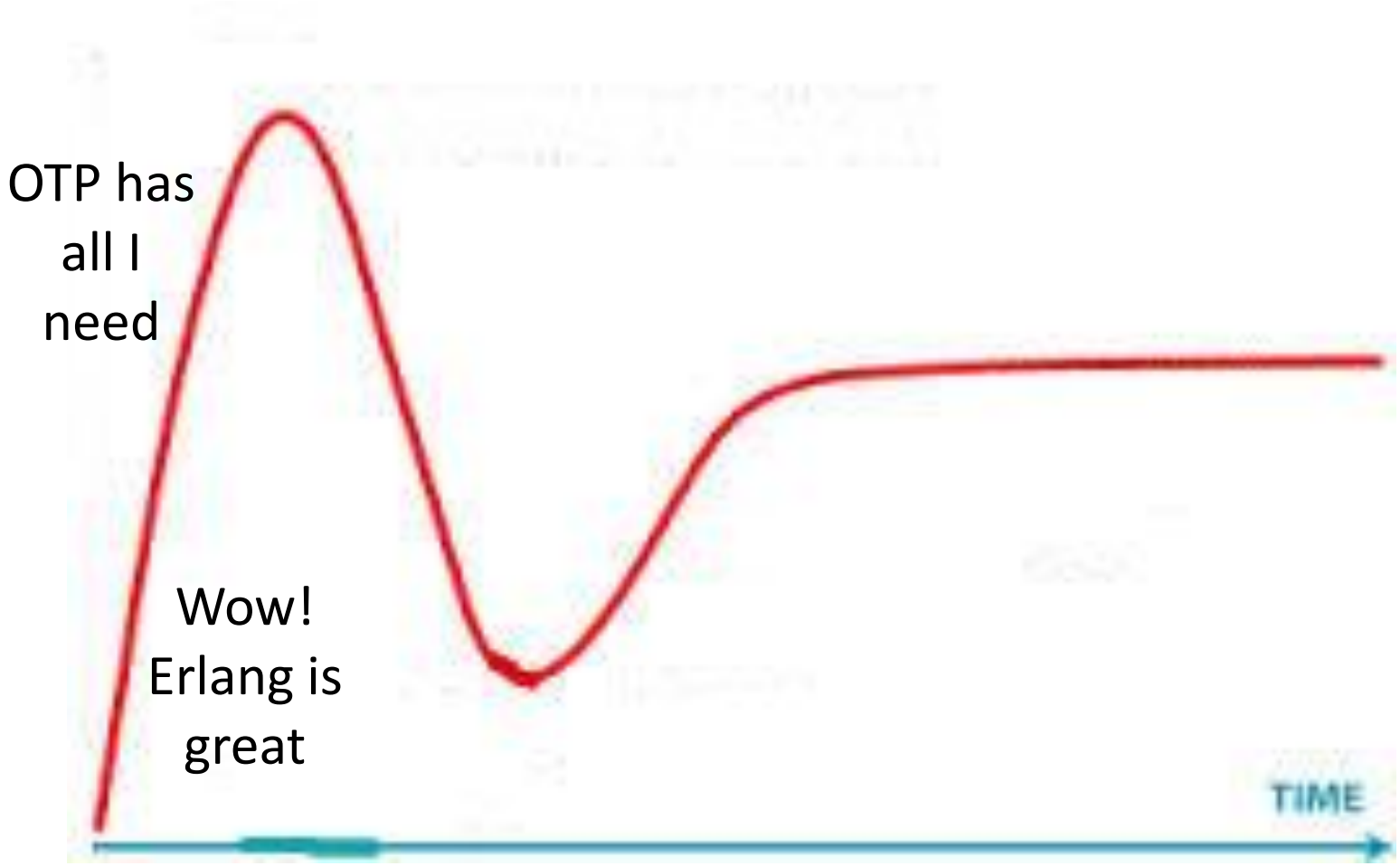
GO BIG

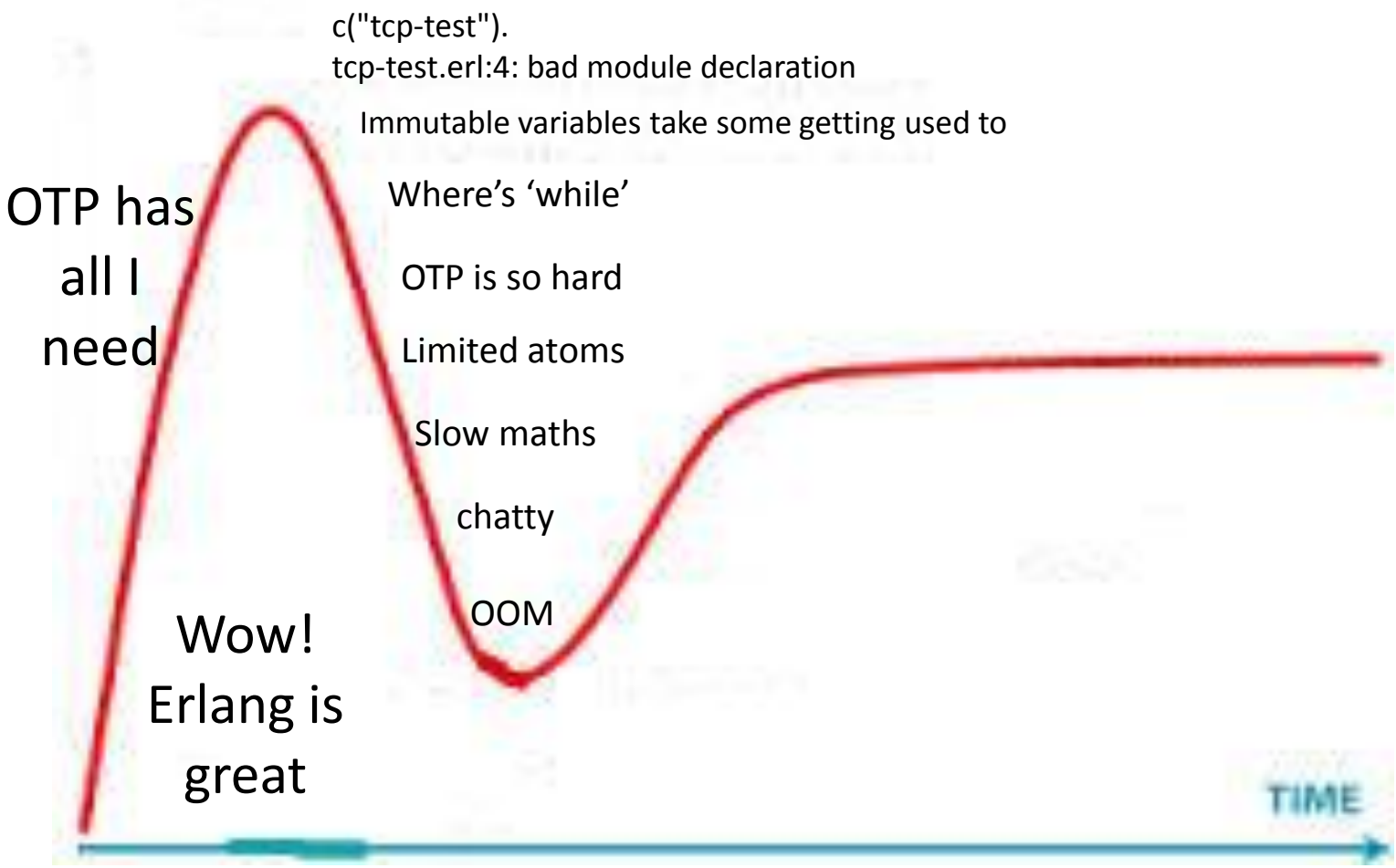
- SCALING ERLANG

Richard Croucher

- Platform architect
  - Chief Architect at Sun Microsystems where I helped create the dotcom deployment standard and designed and deployed a 1024 server cluster
  - Principle DevOPs Architect at Microsoft for all its Internet properties . Adding 4000 servers a month. Established the dynamic computing working group to create design patterns for Cloud computing
  - Primary focus over the last decade has been High Frequency Trading systems for Banks - pushing technology to extremes
  - Involved with several Cloud startups
- Code in multiple languages
  - Transitioned through Assembler, Pascal, C, C++, Java , C# and Erlang
- Love new technology and finding novel ways to use existing technologies
- Degrees in Physics, Electronics and Materials Science
- Elected Fellow of the British Computer Society and Fellow of STAC Research
- See www.informatix-sol.com







c("tcp-test").
tcp-test.erl:4: bad module declaration

Immutable variables take some getting used to

OTP has all I need

Where's 'while'

OTP is so hard

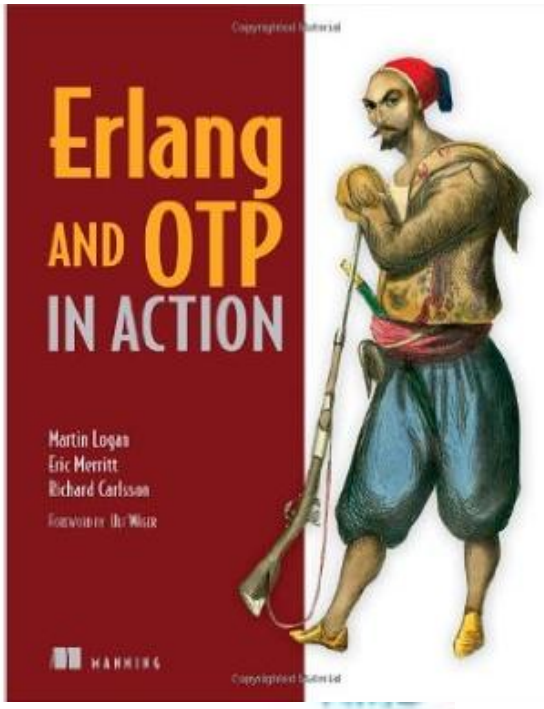
Limited atoms

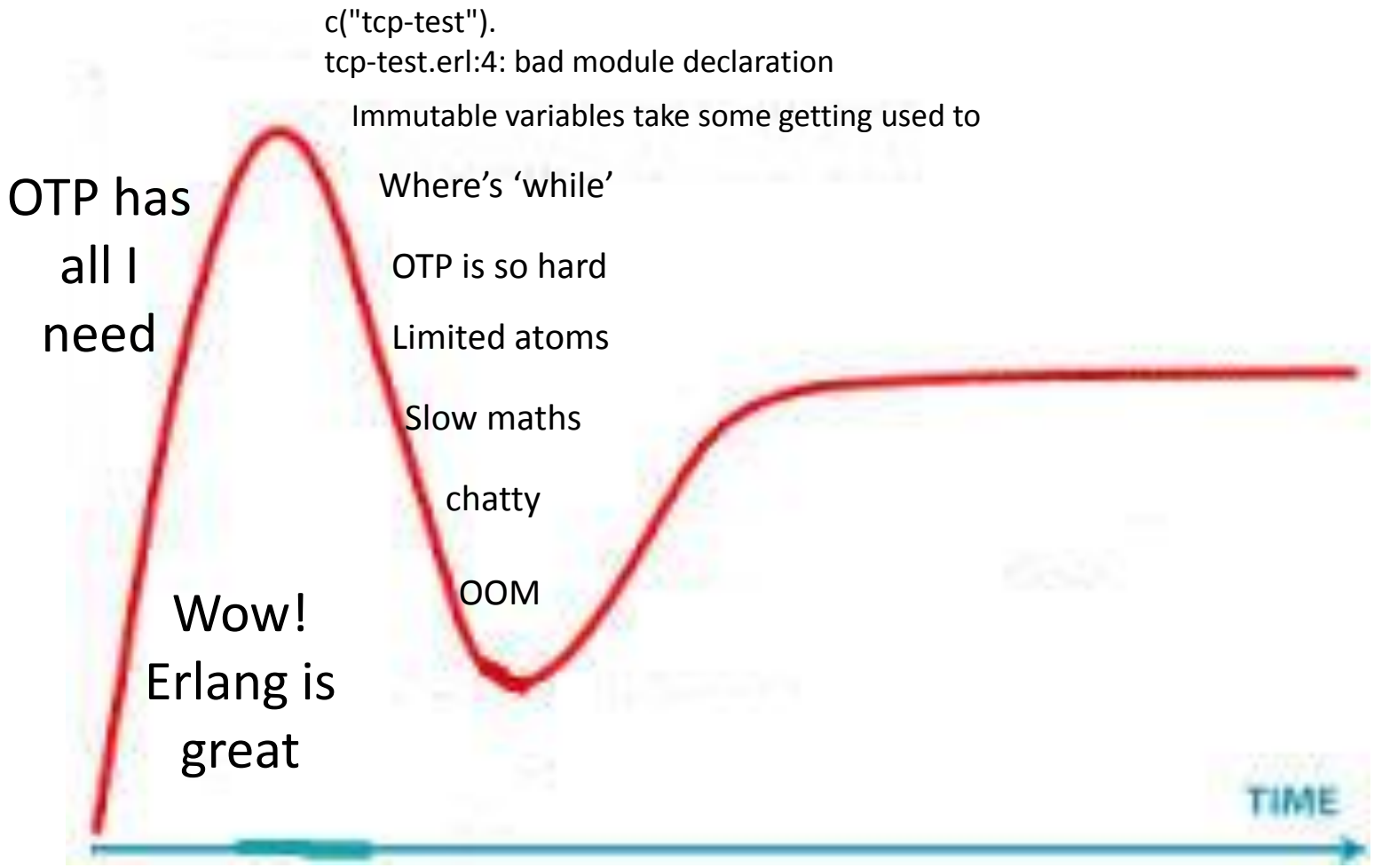
Slow maths

chatty

OOM

Wow! Erlang is great





- Single computers that fill a room
- Continual race to build the biggest - www.top500.org
- Solving big science challenges
  - global warming, weather forecasting, Higgs Boson, nuclear simulations, web Search
- 3 generations of Grid computing:
  1. Distributed Resource Managers
  2. MPI
  3. MapReduce

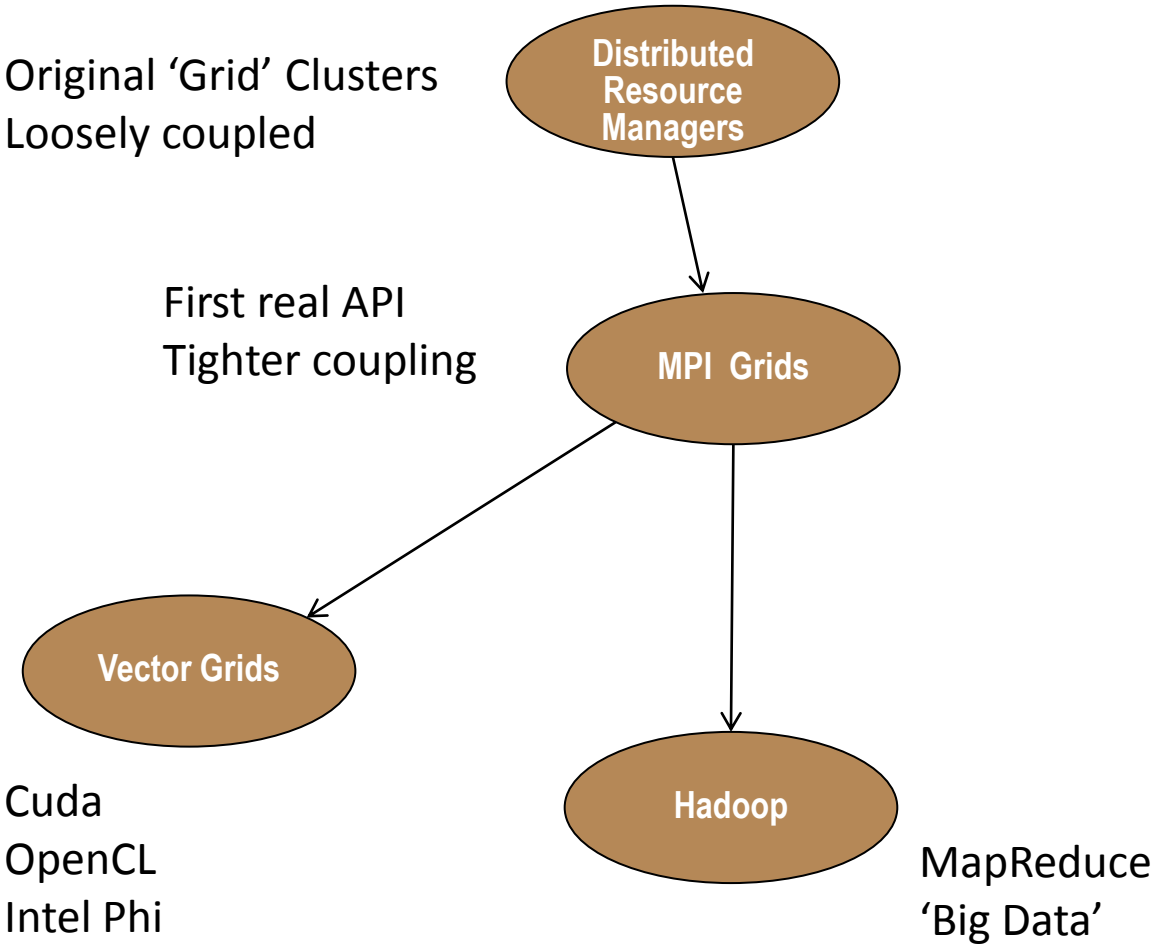
Blue Gene computer, Lawrence Livermore Laboratories USA

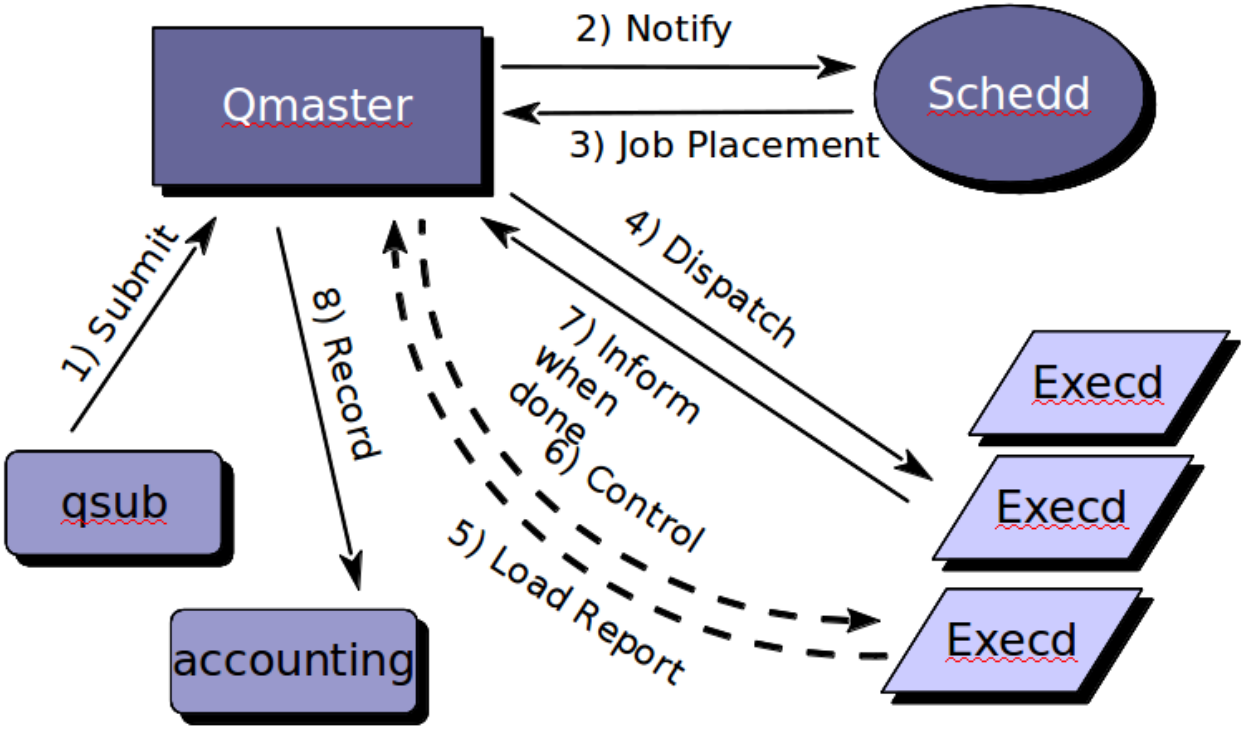
- Single computers that fill a room
- Continual race to build the biggest - www.top500.org
- Solving big science challenges
  - global warming, weather forecasting, Higgs Boson, nuclear simulations, web Search
- 3 generations of Grid computing:
  1. Distributed Resource Managers
  2. MPI
  3. MapReduce

Blue Gene computer, Lawrence Livermore Laboratories USA

Original focus on pure compute but as datasets (TB-PB) grew emphasis changed to moving data to the compute elements. The methods uses to accomplish this are loosely referred to as 'Big Data'

Dissatisfaction with cost and performance of big SMP servers





- Jobs are submitted and then queued to start on the required number of matching servers, each with their own execd
- Jobs are essentially scripts with some manual partitioning of the data set
- Typically used NFS to share data between nodes
- Examples - Condor, Sun Grid Engine, Platform, TIBCO Datasynapse ...

- Message Passing Interface (MPI) standard
- Grid API in 'C' and Fortran
- Several implementations - MPICH-G2, GridMPI, LAM-MPI
- Supports both synchronous and asynchronous messaging between nodes in a grid cluster
- Enables individual jobs to share data, rendezvous
- Supports multicast for efficient internode broadcasting
- Uses Cluster files systems such as Lustre, GPFS, Pananas to provide high performance access to shared data
- Top500 rating is via Linpack benchmark – Fortran benchmark, massive array manipulation

Sensitivity of MPI performance to network latency led to tuning and optimization of the network

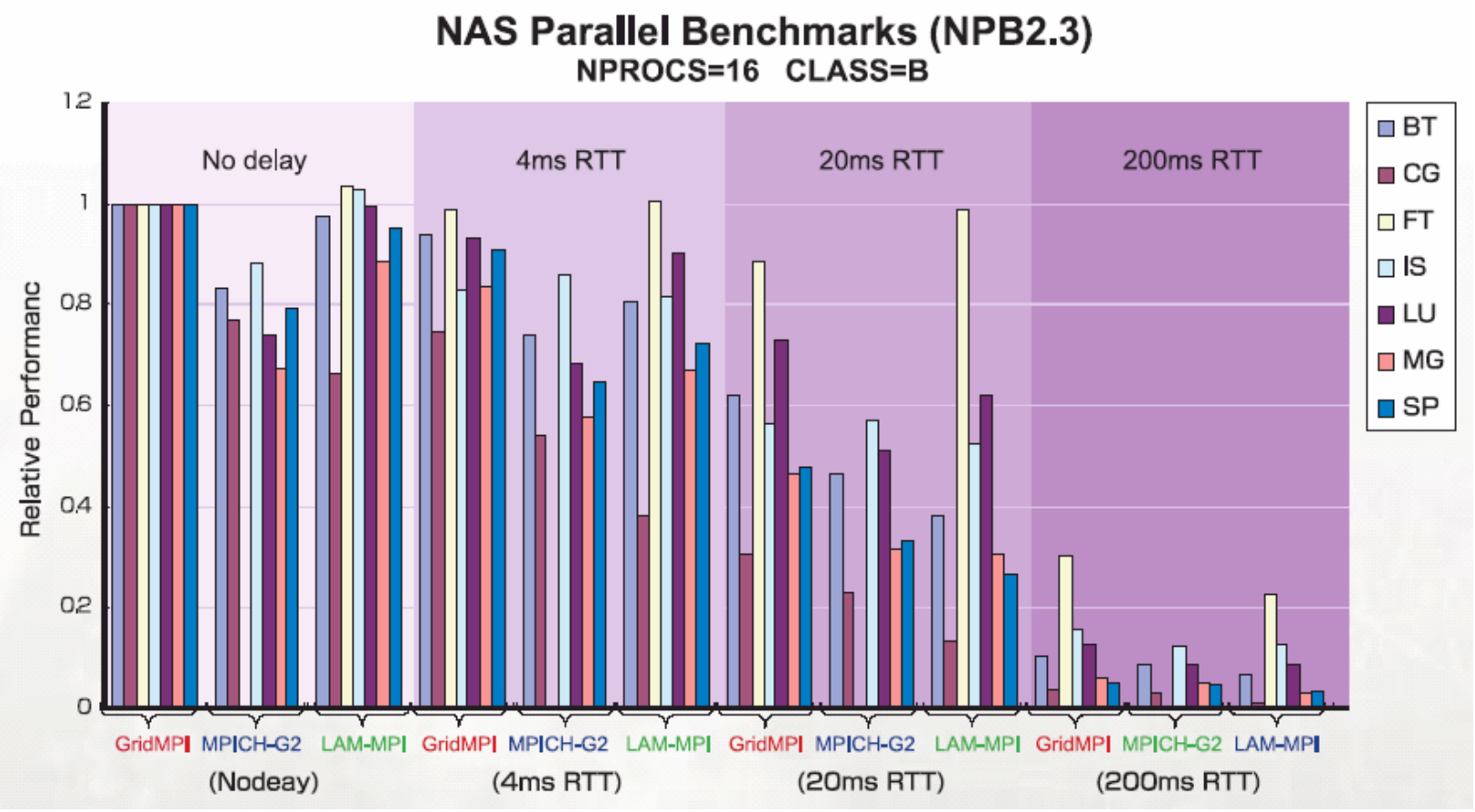
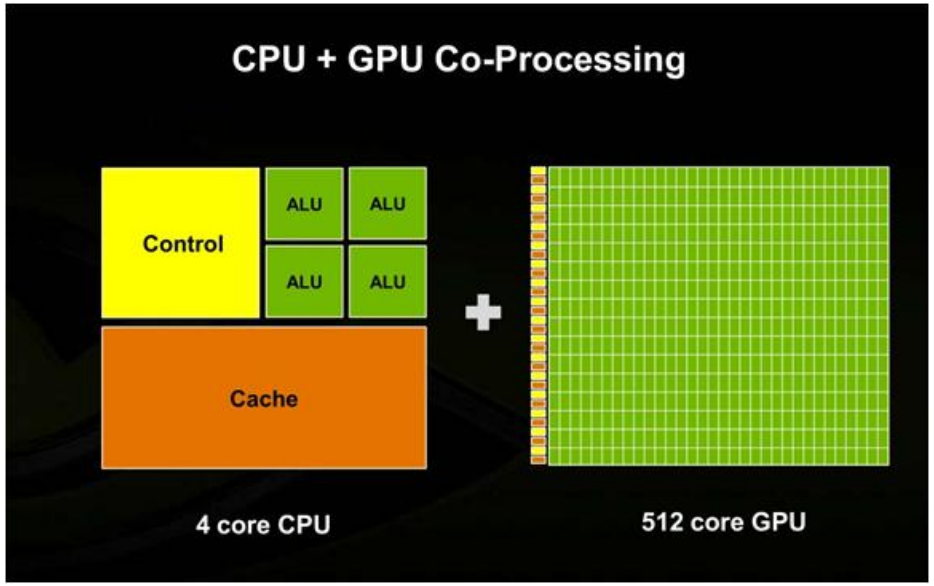


Chart courtesy of gridmpi.org

- Single Instruction Multiple Data are a type of computer specifically designed for parallel computing
  - Same program executes in parallel across different slices of the dataset
- Used in GPU's since visualization suits SIMD architectures
- Provide access to large numbers of floating point units so good for vector intensive workloads
- GPU consumerization drove down prices resulting in people adding cards and using graphics primitives to program them
- Nvidia commercialised their GPU's into 'Physics' processors and defined a generalized API - CUDA – 'C' API
- ATI/AMD followed suite with their own GPU's and programming languages
- Industry standardization followed with OpenCL
- Intel launched 'many cores', aka 'Phi' which provides 64 PentiumP4 cores + additional vector registers per chip

- 2U server with 4x Tesla Nvidia cards + 2x Intel x86
- 4 off Tesla's each with :
  - 512 cores
  - 665 Gigaflops double precision
  - 1331 gigaflops single precision
  - 177 GB/sec memory B/W
- 2 off x86 CPUs with:
  - 20 cores
  - 256GB memory

Total 2068 cores per 2U server, or 41K cores per rack



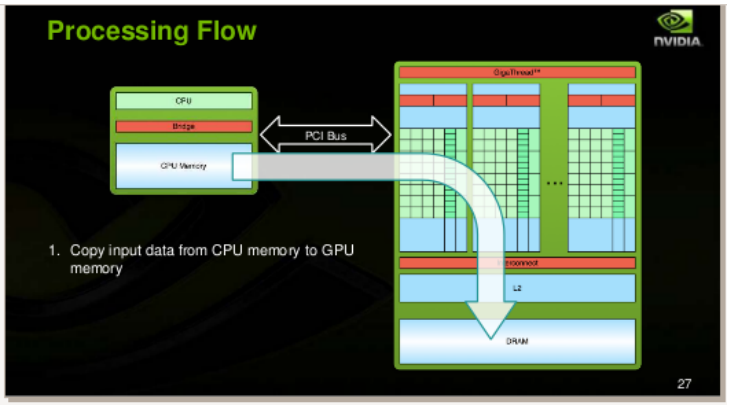
Need to match CPU:GPU ratio to application profile

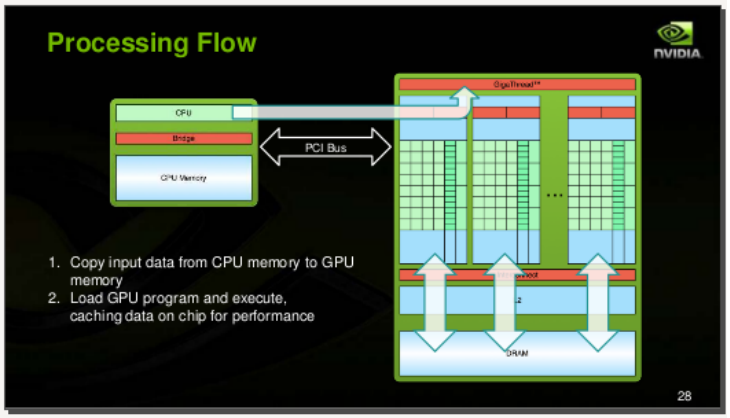
1. Copy input data from CPU memory to SIMD memory

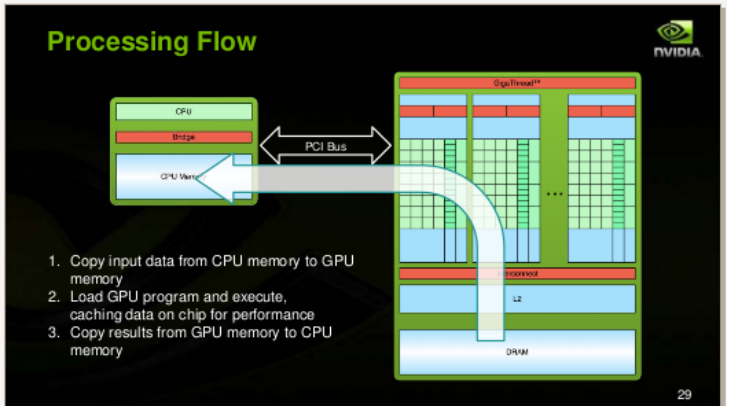
2. Load program code onto SIMD. Note: same code executes on each thread

3. Start program execution and wait until finished

4. Copy results back from SIMD and merge results







TOP10 November 2013

1. **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P
   NUDT
2. **Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x
   Cray Inc.
3. **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom
   IBM
4. K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
   Fujitsu
5. **Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom
   IBM
6. **Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x
   Cray Inc.
7. **Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P
   Dell
8. **JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect
   IBM
9. **Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect
   IBM
10. **SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR
    IBM

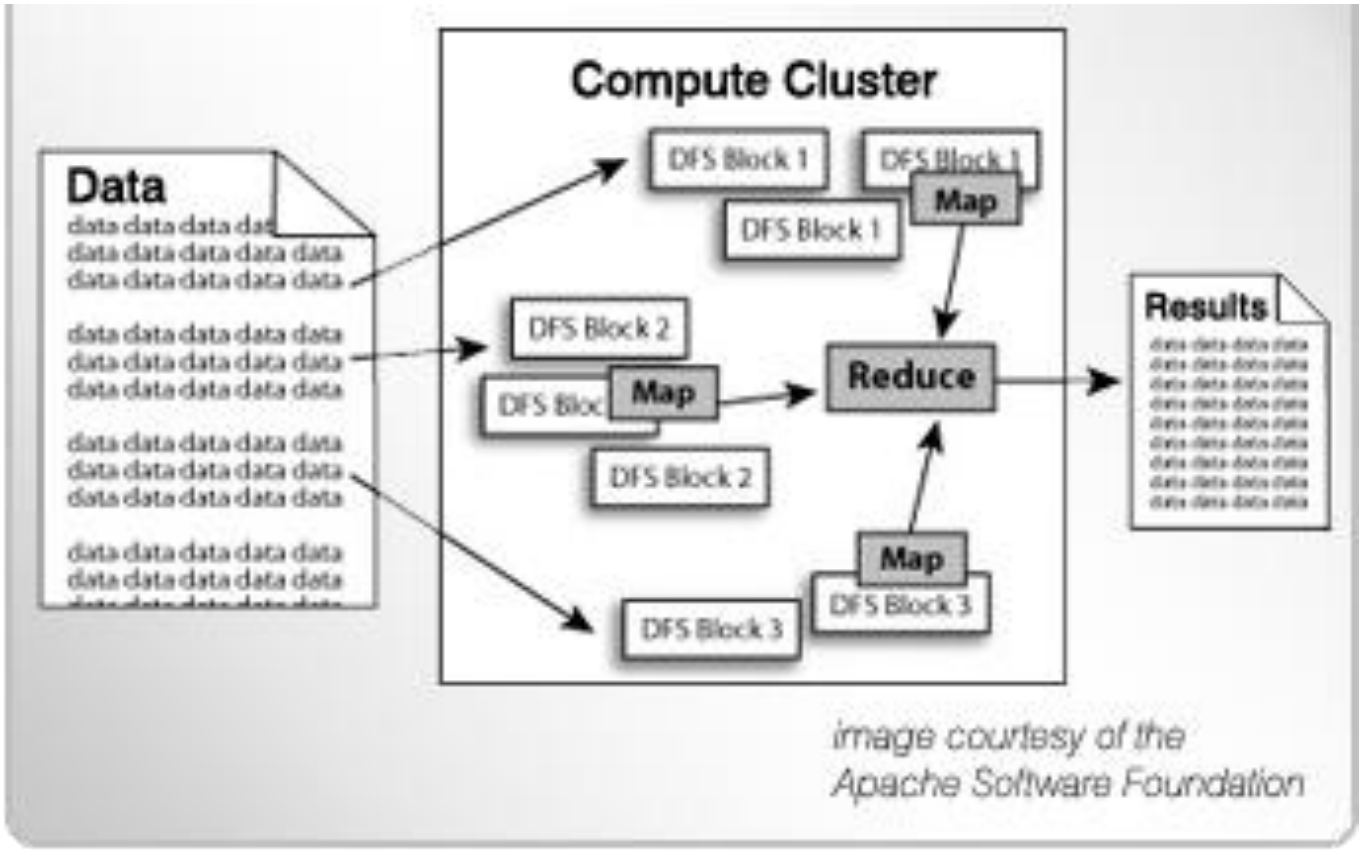
- Tianhe-2 (MilkyWay2)
- National Super Computer Center Guangzhou
- Intel E5-2692 + Xeon Phi
- Total 3,120,000 cores
- 33,863 Tflops Rmax

- Most Users want to achieve the highest compute power for a given budget and/or MWatts
- Design the software to cope with failure
  - Avoid any Single Points of Failure (SPOFS)
- Once the software can survive individual failures, then cheaper commodity servers can be used
  - e.g 98% of 1500 servers, is better than 99.9% of 1000 servers
- Once there are too many servers to plug into a single network switch, the network design becomes critical
  - Achieving Bandwidth and latency at scale is the challenge
  - 1500 servers on a high b/w, low latency network will typically outperform 2500 servers on a slower network

- Financial Services
  - Options pricing
  - Market valuations
  - Value at Risk - Monte Carlo
- Oil and Gas
  - Seismic analysis
- Media
  - Rendering and visualization
- Engineering
  - 3D design and modelling
  - Crash test
- Health and Pharma
  - DNA
  - Molecular modelling
- Big Science
  - Large Haydron Collider (LHC) results analysis
  - Satellite imaging
  - Weather forecasting
  - Global warming
  - Nuclear simulation

Popularized in a paper published by Google.

http://research.google.com/archive/mapreduce.html



- The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

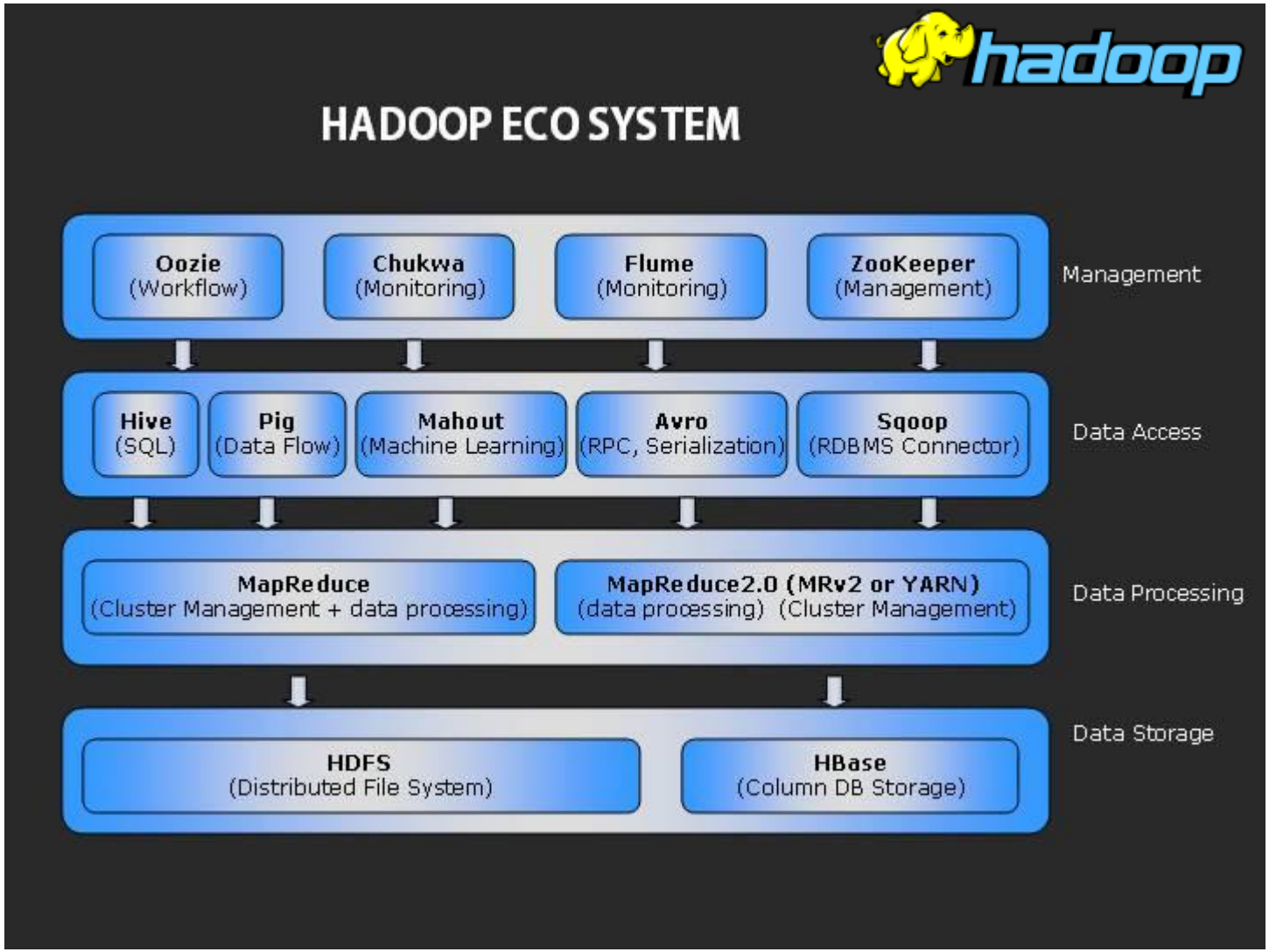
  Map(k1,v1) -> list(k2,v2)

- The Map function is applied in parallel to every item in the input dataset. This produces a list of (k2,v2) pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, thus creating one group for each one of the different generated keys.
- The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

  Reduce(k2, list (v2)) -> list(v3)

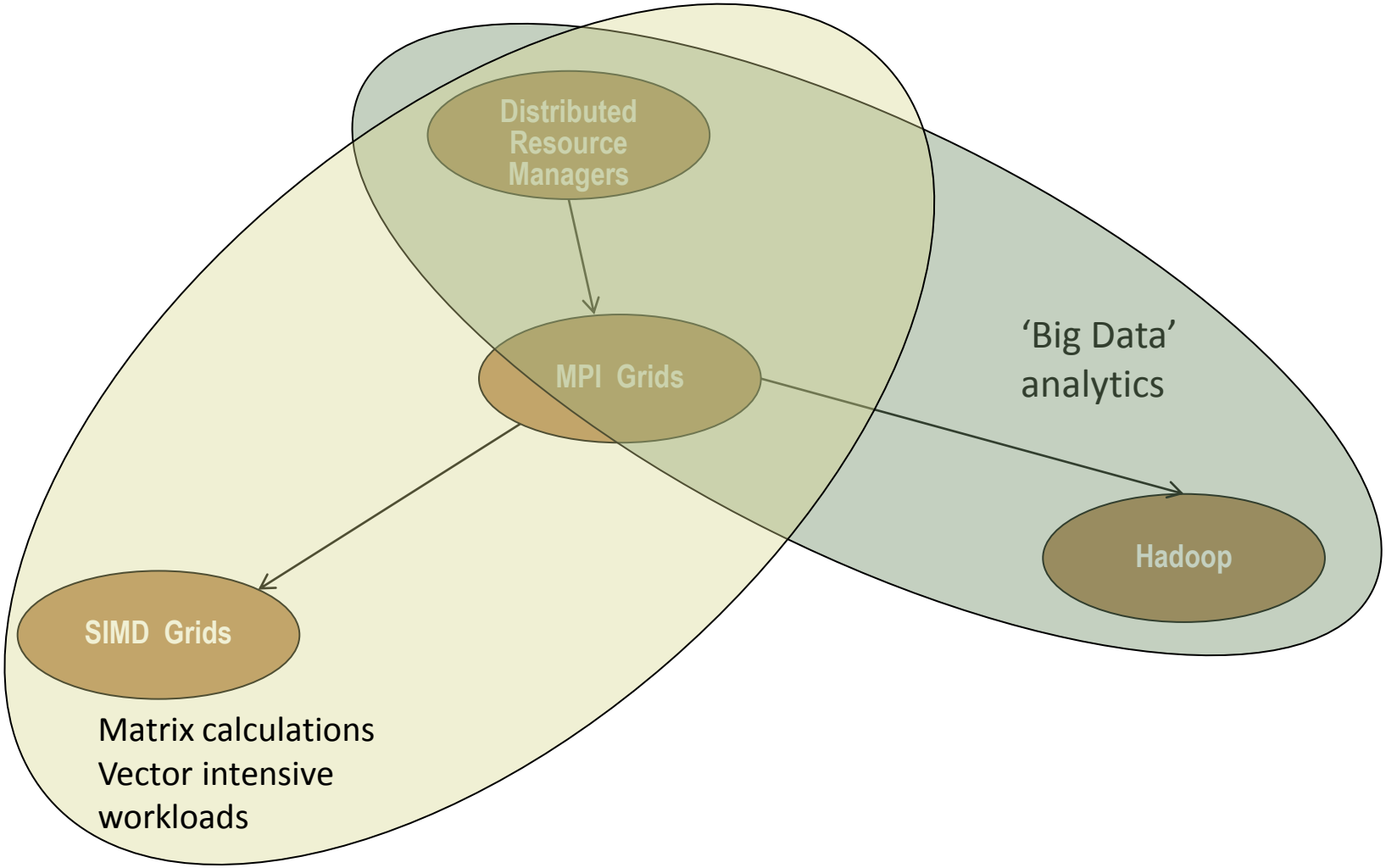
- Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

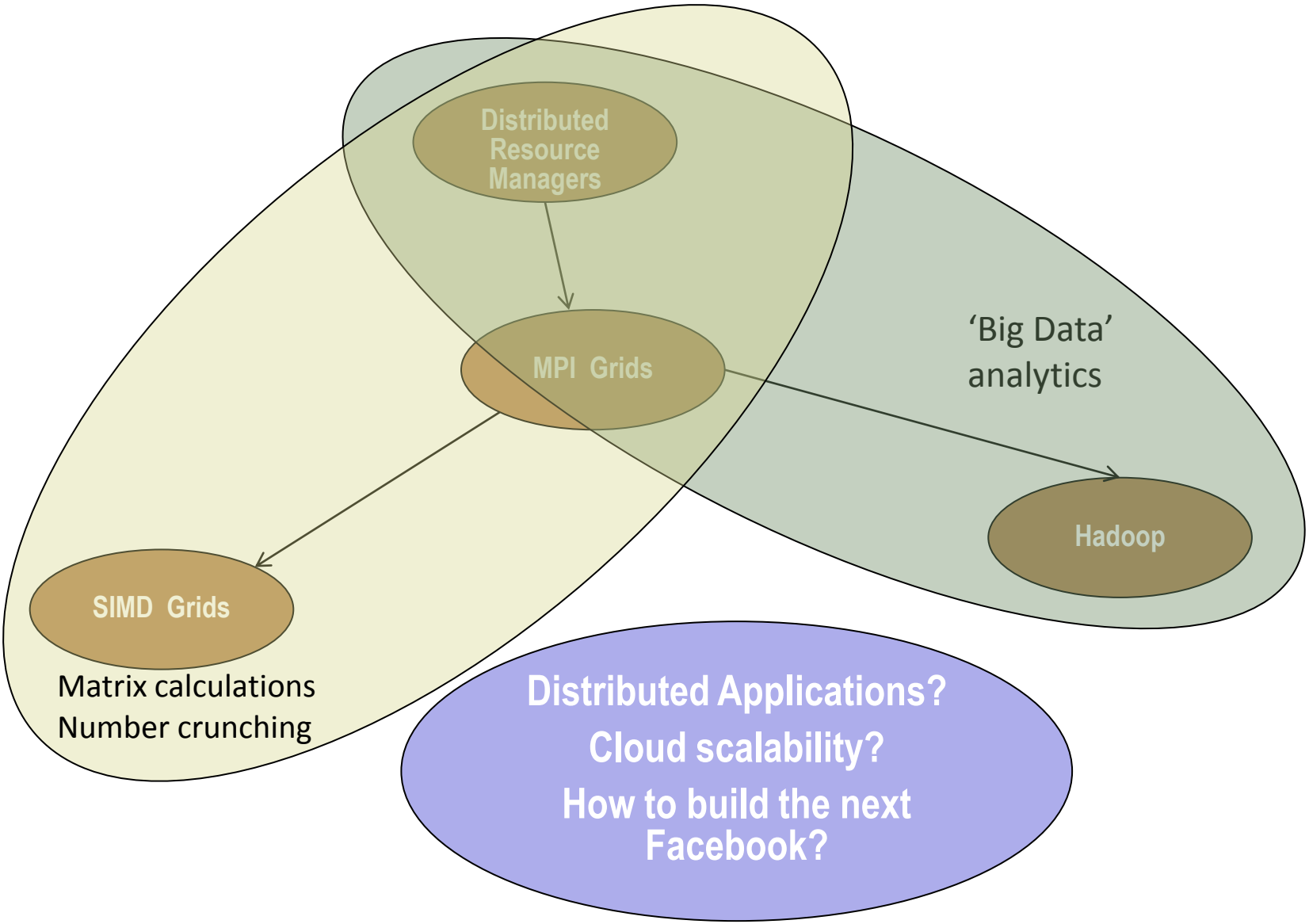
- Doug Cutting, then Search Director at the Internet Archive and Mike Cafaella had built their own Search indexer called Nutch in 2002-2004. This could crawl hundreds of million of web sites but needed a lot of hand holding and raised concerns as to whether it could scale.
- When Google published its papers on Google File system (Oct 2003) and later with MapReduce (Dec 2004). Doug released this was a better approach . They implemented these in Java and added Nutch on top.
- In 2006, Doug Cutting joined Yahoo. They'd also been impressed by Google's papers. They spun out the storage and processing components Doug had been working on to create the open source Apache Hadoop project, keeping Nutch as an internal project.
- The initial code only scaled to 20 servers, but Yahoo committed heavy resources and eventually scaled it to the thousands of servers necessary to run their web indexing
- Yahoo created a research grid internally and the data scientists using it discovered they could use it for analytics - this is the principle use case Hadoop is used for today.
- In 2011 Yahoo's Hadoop environment was reported to be 42,000 nodes and hundreds of Peta bytes of data.
- Facebook are probably the largest Hadoop user now, in 2012 it was reported to be 100PB, growing at ½ PB per day
- Google don't use Hadoop, their MapReduce implementation is written in C++

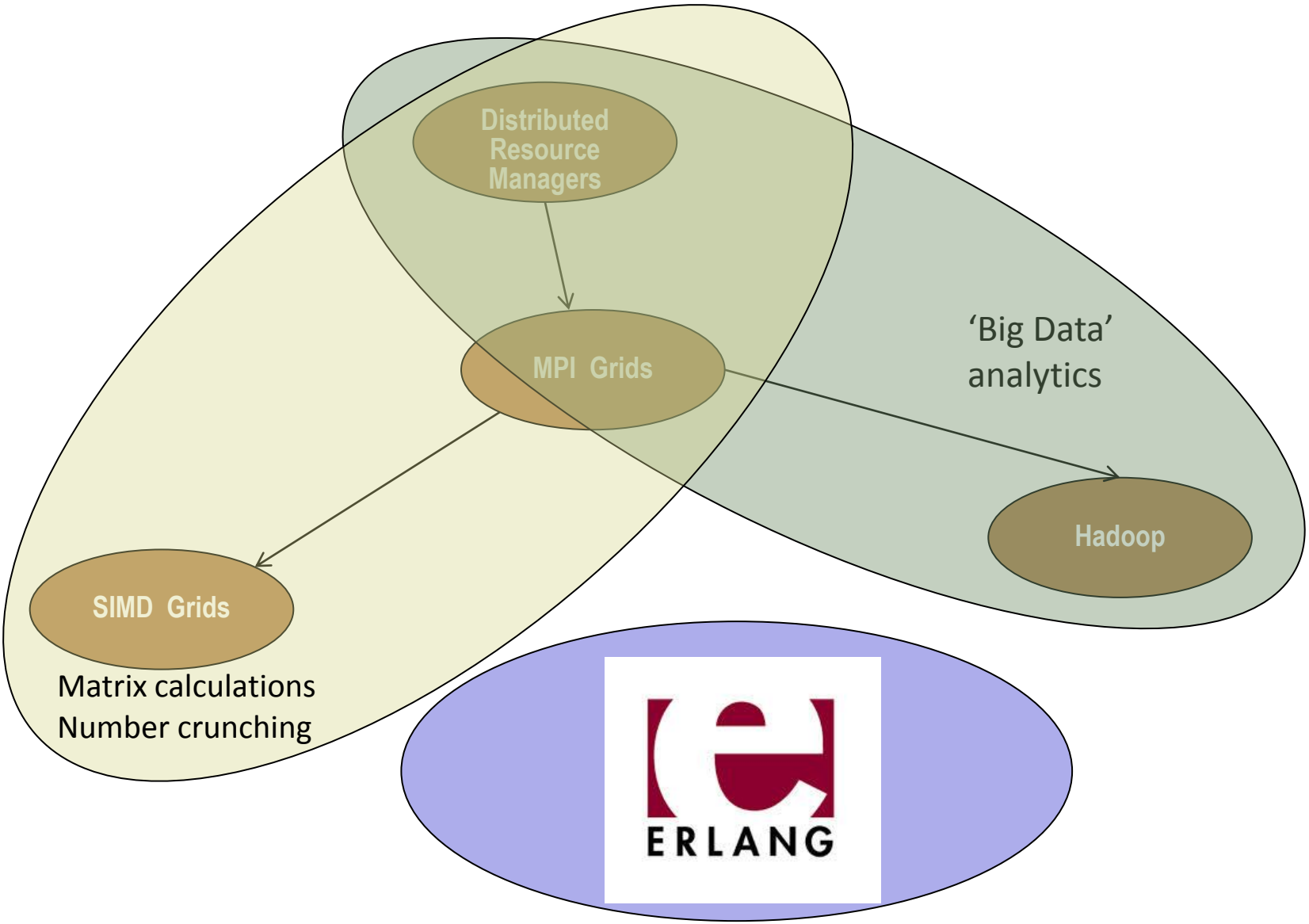


Courtesy of Bala Sundaram at http://futureanalytics.blogspot.co.uk/

- Hadoop itself only provides basic get/set/scan primitives. Add on Query Engines are used to supplement this
  - Hive - original Apache project implementation for Hadoop, SQL like
  - Facebook Presto – recently Open Sourced, ANSI-SQL, suitable for interactive queries
  - Cloudera Impala Query language
  - Dremel – Open source implementation of Google BigQuery API
- Many Hadoop clones which are compliant with the same set of APIs - Cloudera and other adding value to the original code such as IBM and Intel
- Derivative projects such as Apache Shark aim to optimize for interactive User cases







- Scalability, particularly concurrency is too hard with imperative programming languages, - functional programs are a better fit
- Erlang designed for concurrency
  - Immutable variables
  - Always (mostly) pass by copy
  - Asynchronous messaging
  - Execute a function on a remote node - spawn(Node, Module, Fun, [Args])
- Easy to create clusters
  - erl -sname hostname -setcookie mysecret
  - Now tell everyone I'm here - net_adm:ping(some_other_node).

Open Telecom Platform (OTP) makes Erlang production capable

- OTP provides many required services and generalised supervisor patterns which support the Erlang principle of crash early, automatically restart
- Includes standard behaviours
  - gen_server
    - start_link(ServerName, Module, Args, Options)
  - gen_fsm
    - Finite State machine
  - gen_event
    - Event machine
  - Supported by other components such as Mnesia
    - Distributed, database, on disk and/or in memory
  - Includes Release control, Edoc, live updates, profiling, code inspectors etc.

But..

- Erlang cluster scalability limited due to chattiness, even with hidden nodes
- OTP designed for the specialised compute environment of telephone switches, documented as being limited to 50-100 nodes
- OTP supervision trees are designed to work on local nodes
  - A supervisor is recommended to only supervise local processes due to concern about the reliability of the network
- OTP Distributed Application controller - dist_ac
  - distributed = [{Application, [Timeout,] NodeDesc}]
    - Where NodeDesc is a list of Nodes this application may execute on, in priority order
  - For this to work, first the nodes must contact each other
    - sync_nodes_mandatory = [Node] - all these nodes must be running
    - sync_nodes_optional = [Node] - this nodes may be running
  - Design goal is to keep the application running (on one node) in an environment where its assigned node(s) may fail - does not provide scalability, or even load balancing
- gen_server_cluster (Erlang Central)
  - enables multiple servers to provide a single service, each running on a different erlang node. Implementation it to have one active server out of the cluster, sharing state to the others to they can elect a new leader if it goes down. Implementation does not provide scalability

.. Surely somebody has solved this?

- Computerl - pipeline based compute scheduling.
  - Used initially to automate CT
  - No commits in last 3 years.
  - Unable to get response from Michal Ptaszek at his published email address
- Nodefinder
  - Uses multicast to discover other nodes in the cluster.
  - Separate version available to use on AWS

- Disco project see http://disco.readthedocs.org/en/latest/intro.html
  - MapReduce implementation written in Erlang
  - Reference implementation is a 800 core cluster at Nokia Research centre in Palo Alto
  - New nodes can be added to a running cluster and jobs on failed nodes will be restarted elsewhere
  - HTTP REST API allows jobs to be submitted in many languages - Python seems the most common
  - Integrated cluster file system – Disco Distributed Filesystem (DDFS). Horizontally scalable. Designed for large stores. Blob storage defaulting to 3 replica's
  - Dependency on Python to run

- Riak_core
  - A proven component used in Riak noSQL and separately available
  - Node watcher to manage cluster membership, includes API to advertise and discover specific services. Enables adding and removing nodes
  - Master/worker pattern using vnodes as workers.
  - Stores cluster global state in the (Dynamo) ring by 'gossiping'
  - Which node to use depends on hashing within request ala Dynamo, filters out down nodes, relies in ability of any node to service request

  - Can be used generically but does have a strong Riak bias

Release

- A EU funded program with collaborators including Uppsala University, Herriot Watt University, University of Kent, Erlang Solutions, Ericsson, University of Glasgow, EDF
- Goal is to evolve Erlang to run on 100K core environments
- Carrying out projects to improve scalability of the Erlang VM, defined a distributed component Ontology, prototyped a cloud deployment tool (Wombat)
- Current (May 2014) software available:
  - benchErl - benchmarking tool, Dialyzer, Percept2, ErLLVM

White papers talk about:

- s_groups will allow partitioning of a cluster and reduce chatter.
- Semi-explicit placement, replacing the default round-robin placement, placement hints to encourage deployment in same node for example.
- load management - plan of offer a load server, will collect load information and decide where to spawn a new process. Will have one load server per node, chose_node function

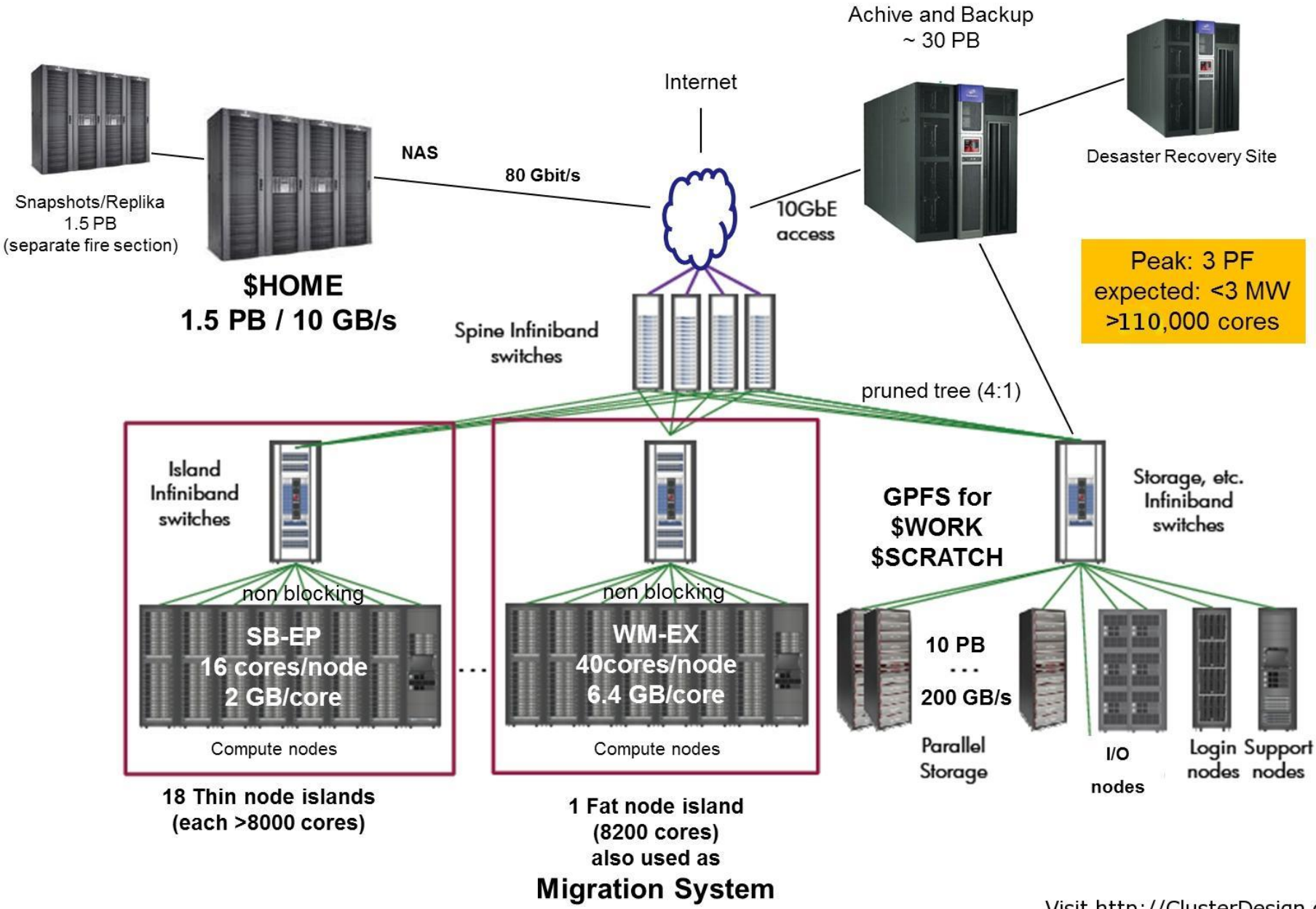
- Erlang/OTP together help you solve the most difficult part of creating massively scalable applications – writing scalable code which can run reliably
- Lots of good tech that can be leveraged but still a development to be done
- Longer term – RELEASE will help enormously but don't think we can wait that long
- Lots going on outside the Erlang community that could also be leveraged e.g.
  - RDMA to overcome the TCP/IP bottleneck
  - Multicast to reduce chatiness
- It's possible to write the next 'Facebook' in Erlang/OTP and there are lots of people all working independently to achieve cloud scale
- Real opportunity is for Erlang/OTP to take the lead as the premier Cloud platform
  - harnessing all these separate activities
  - leveraging lessons learnt elsewhere building massive scale

Richard.Croucher@informatix-sol.com www.informatix-sol.com

SPARE SLIDES

Big Cluster network Example

Informatix Solutions



Visit http://ClusterDesign.org