*Invited Presentation to the 2007 Workshop on the Frontiers of Extreme Computing:*

# Operating Systems for Exascale Computing and Beyond

## *When there are too many cores to count*

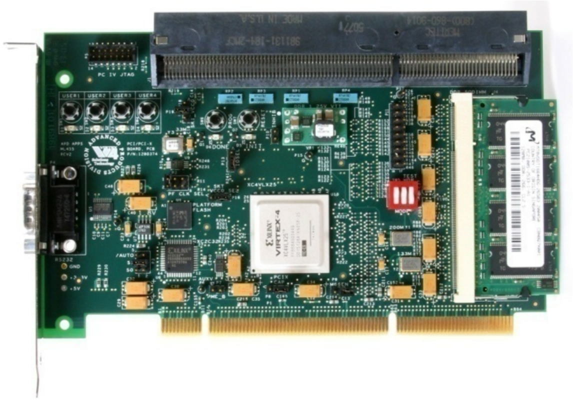
Dr. Thomas Sterling

Faculty, *LSU Center for Computation and Technology*
Arnaud & Edwards Professor, *LSU Department of Computer Science*
Adjunct Professor, *LSU Department of Electrical and Computer Engineering*
Faculty Associate, *California Institute of Technology*
Distinguished Visiting Scientist, *Oak Ridge National Laboratory*

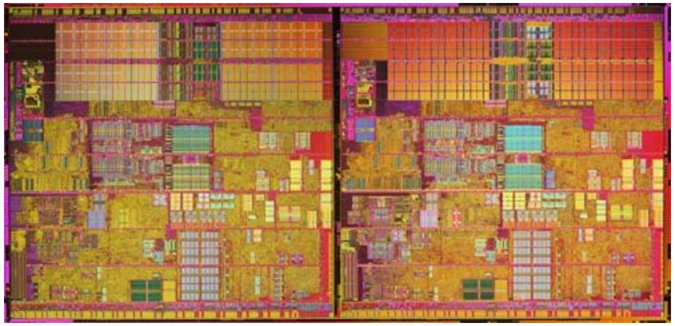
October 22, 2007

cct
Center for Computation & Technology
AT LOUISIANA STATE UNIVERSITY

# Exascale Foundation Assumptions



- Extremes of parallelism
  - 100 million cores
  - Billions of threads
  - Additional fine grain parallelism
- Everything is multi-cycle latency
  - Synchronous domains & asynchronous global exchange
  - Clock rates > 20 GHz
  - Pipelined access to register set
  - top level “cache” is 10’s of cycles away
  - Local memory could be 1000 cycles access latency
    - Processor in memory will dramatically reduce this
- Complex hierarchical global name spaces
- Failure modes
  - Rapacious
  - Repeated
- I/O
  - Many to one
  - Virtualizing the persistent storage
  - Streaming interactive



# Exascale OS Requirements

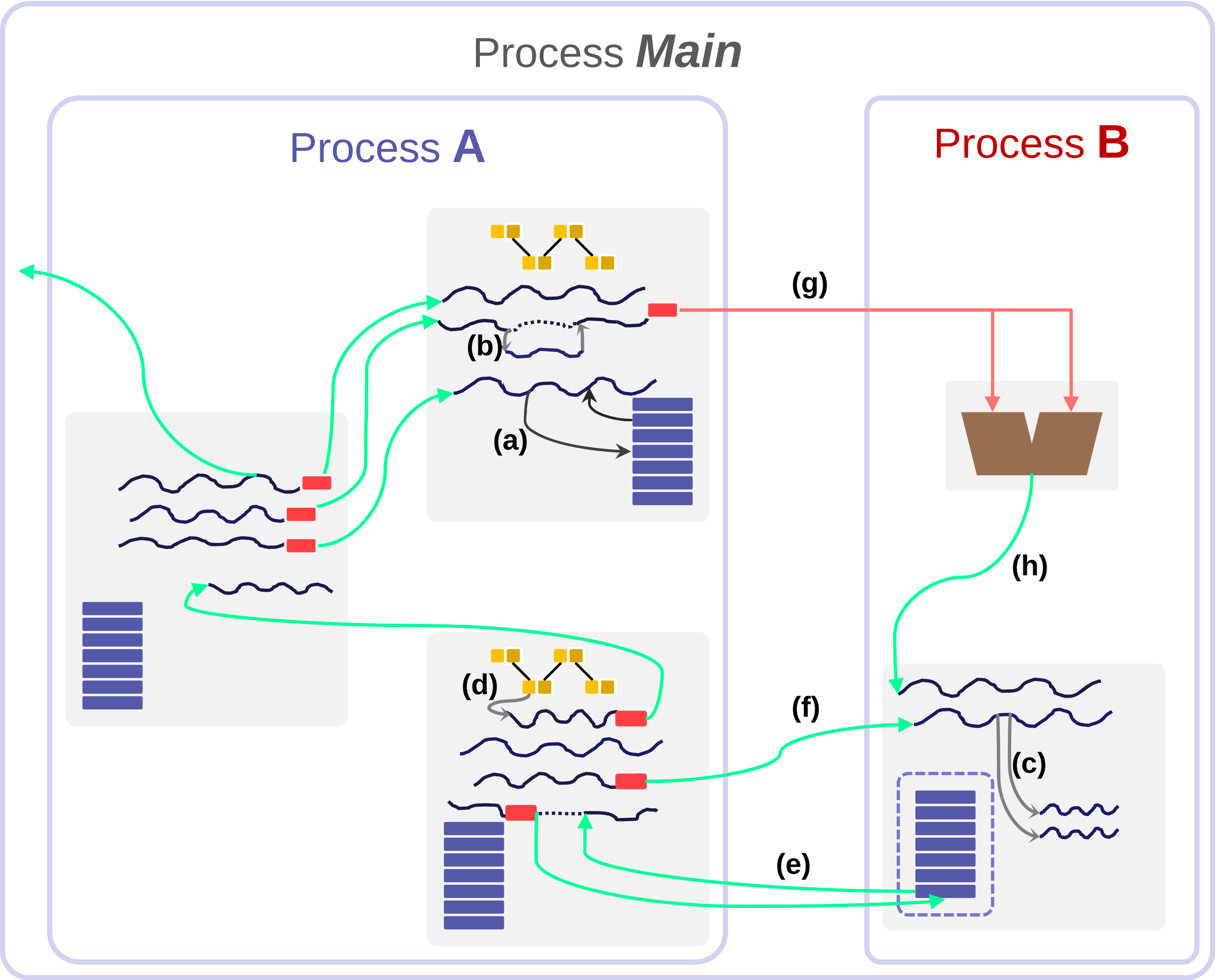
- The Obvious
  - User API services
  - Resource management
  - I/O abstraction
    - User interface
    - File I/O
    - External data streaming
  - Fault recovery and protection
- Keeping the lid on complexity
  - Difficulty of user control should be order constant
  - One environment to rule them all
    - But all *what*?
- Elastic
  - Scalability
    - With effiency
  - Dynamically flexible
    - Take as much as you need, use as much as you take
    - Just in time provisioning
    - Heterogeneity
  - Multi-modal
    - Different policy cultures for different application requirements
    - Coexistence through computational détente

# Where we don’t want to be:

- Garbage piles
  - We avoid it - *because it stinks*
  - Hypervisors on top of
  - Distributed middleware on top of
  - Node Linux’s on top of
  - Core services
  - What’s a compiler to do!?
- Dim witted
  - Lightweight shouldn’t mean: “lobotomized” kernels
- Big Brother – virtual machine
  - Don’t worry your silly little head about details
  - The “Smile, be happy” portability mindset

# Core Trek (the next generation?)

In Exascale Computing: *Space* **is** the "final frontier"



### Legend

- ▬ - local memory
- ⬚ - virtual pages in PGAS
- 🟥 - PGAS address translation
- ▼ - accelerators
- - LCOs
- → - (green arrows) Parcels
- ∿ - threads
- → - function invocations
- → - local load-store operations
- ░ - Locality (light gray shaded areas)

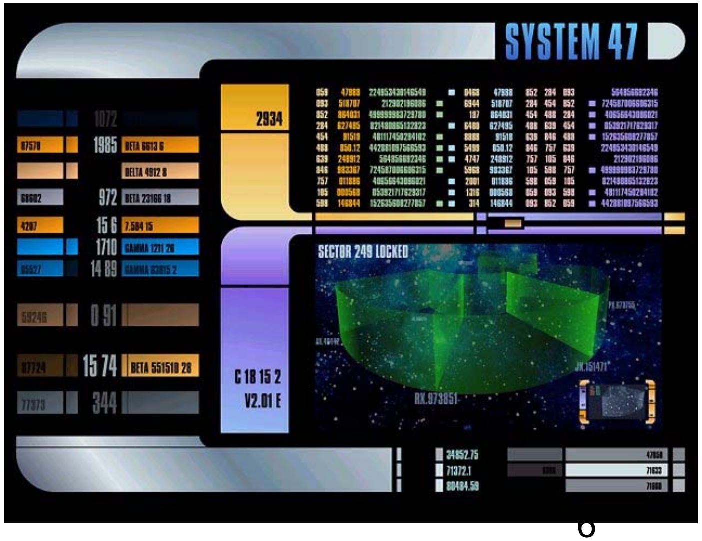
### Key Activities

**(a)** – Local data access
**(b)** – local thread invocation (co-routine)
**(c)** – local thread invocation (concurrent threads)
**(d)** – LCO spawning a thread
**(e)** – remote atomic memory operation through parcels
**(f)** – remote thread invocation through parcel
**(g)** – percolation
**(h)** – thread creation as a result of continuation action

To Boldly Code, where no thread has *goto*'d before

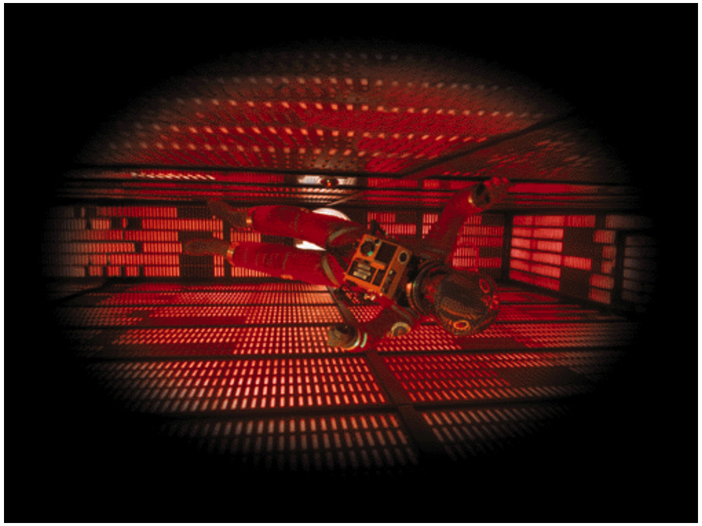
# Federation OS Environments

- When there are too many cores to count
  - Sea of resources
  - Precludes explicit programmer management
- Two layers
  - Lower: Local functional services at hardware resources
  - Higher: global policy and API
- Multi-verse of abstract OS’s
  - Selectable
  - Customizable (on the fly)
  - Concurrent
  - Overlapping



# Sym-biOS for Exascale Computing

- Complexity of operation
  - Not through complexity of design
  - But through complex dynamic interaction of myriad simple functions
  - A property of emergent behavior
- Global services achieved through synergism of local services
  - Local service functions operate within synchronous physical domains
  - Global services achieved through brokered local contributors
- Global Sym-biOS locally interfaced global service layers
  - Memory layer
  - Communication layer
  - Parallel control flow layer



# Paving the Path

- Is the OS community large enough to explore non-typical solutions?
- Do we know enough from hands-on experience with really BIG systems?
- What is the interface that has to be supported to provide a non-disruptive path for existing application code base?
- What are measurable milestones that permit tracking of progress towards realization of future environment?