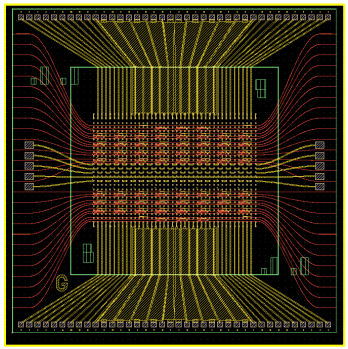
SAND2014-4348C

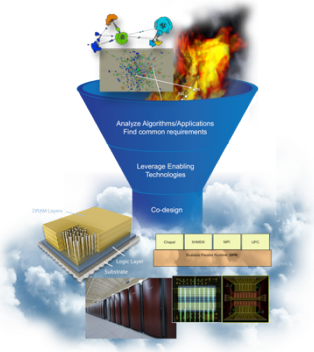
*Exceptional service in the national interest*

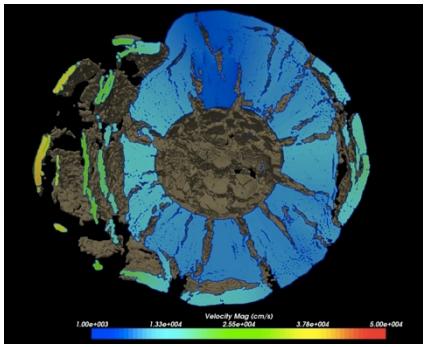
Sandia National Laboratories

XGC

EXTREME-SCALE COMPUTING GRAND CHALLENGE






# Opportunities for Integrating Tasking and Communication Layers

Dylan Stark

U.S. DEPARTMENT OF ENERGY

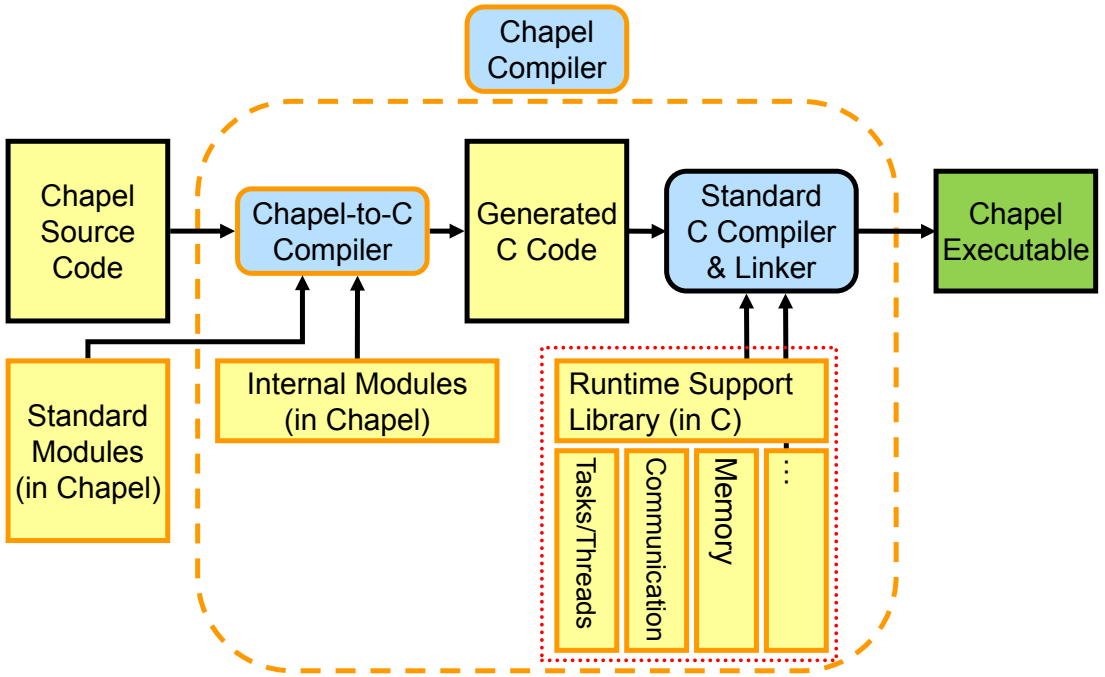
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

# My Objectives for this Talk

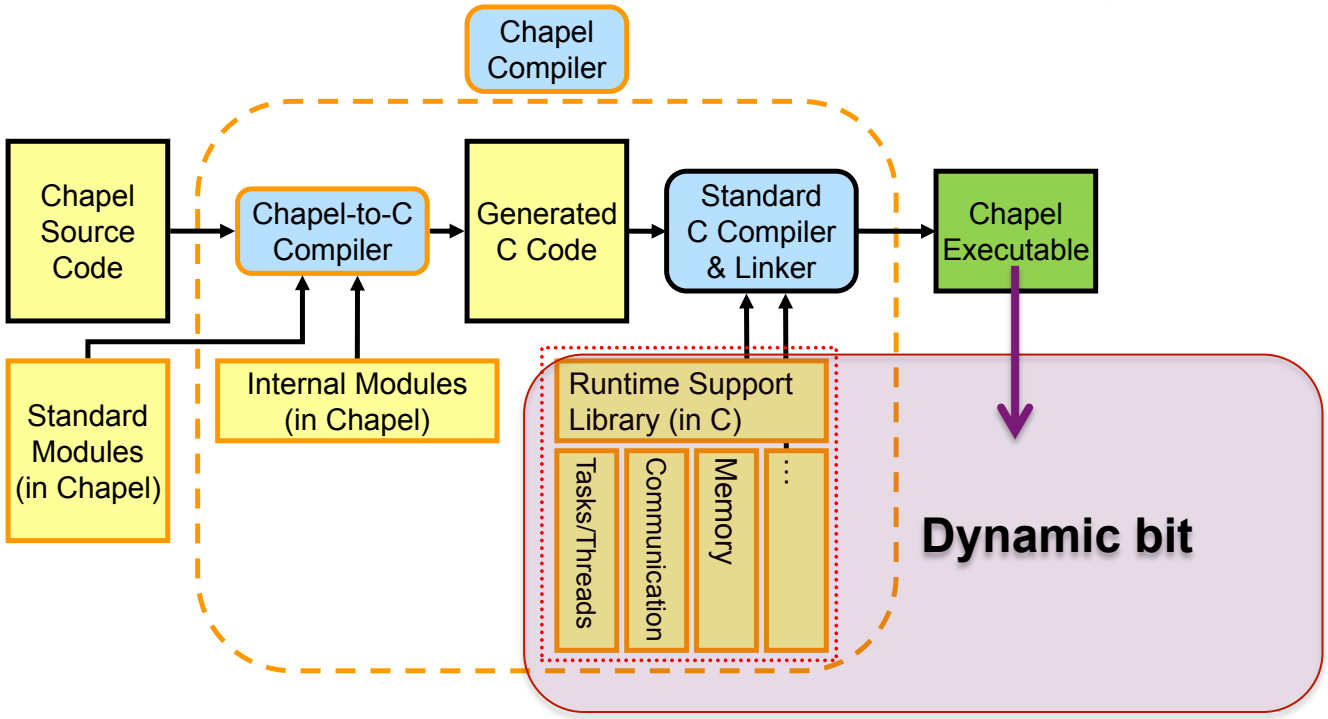

1. Motivate consideration of Chapel at run time
2. Review how Chapel operates over multiple locales
3. Describe our unified runtime attempt
4. See what happens when you put the two together

## Chapel Compilation Architecture



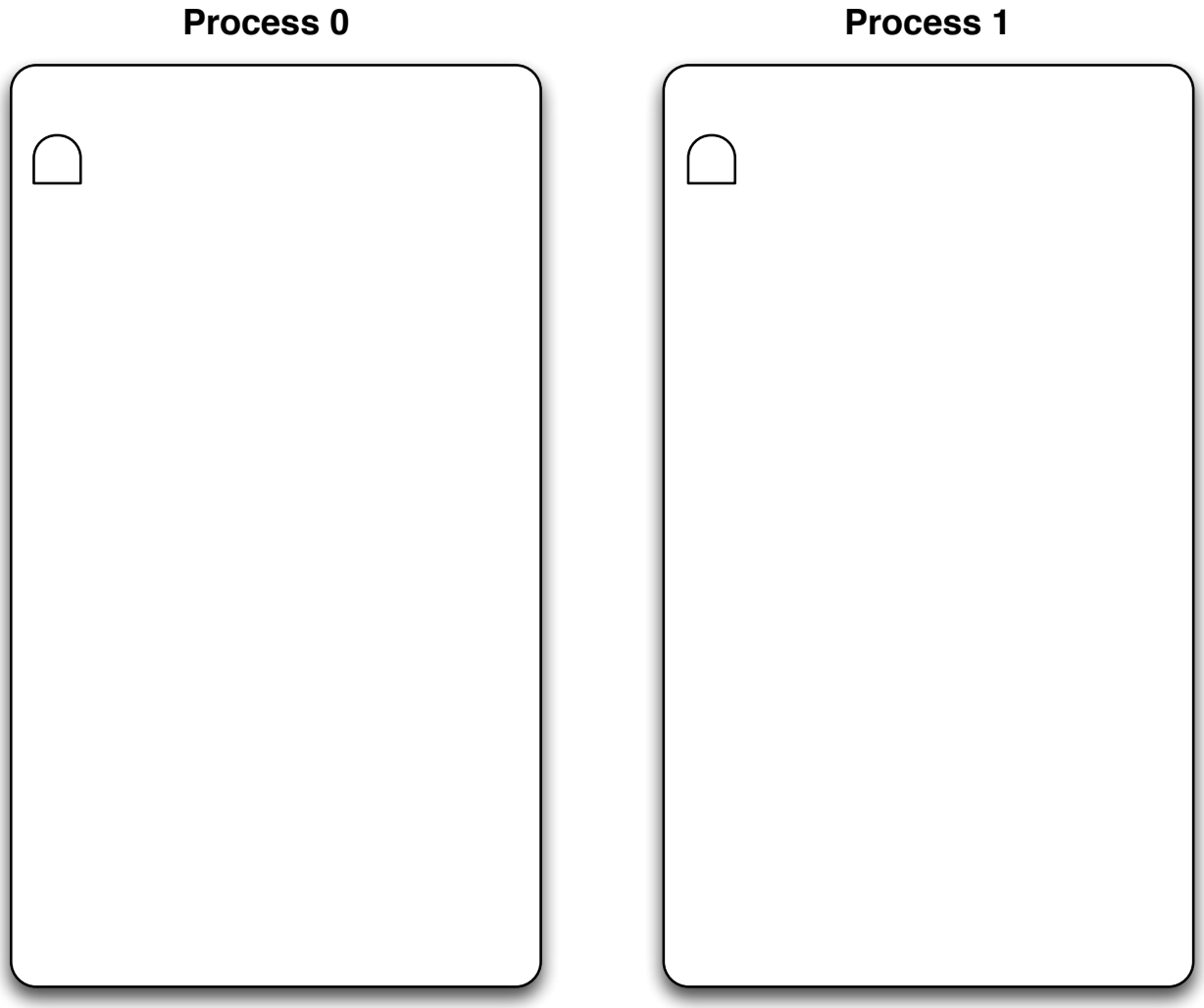

## From the Static to the Dynamic

## Chapel Compilation Architecture



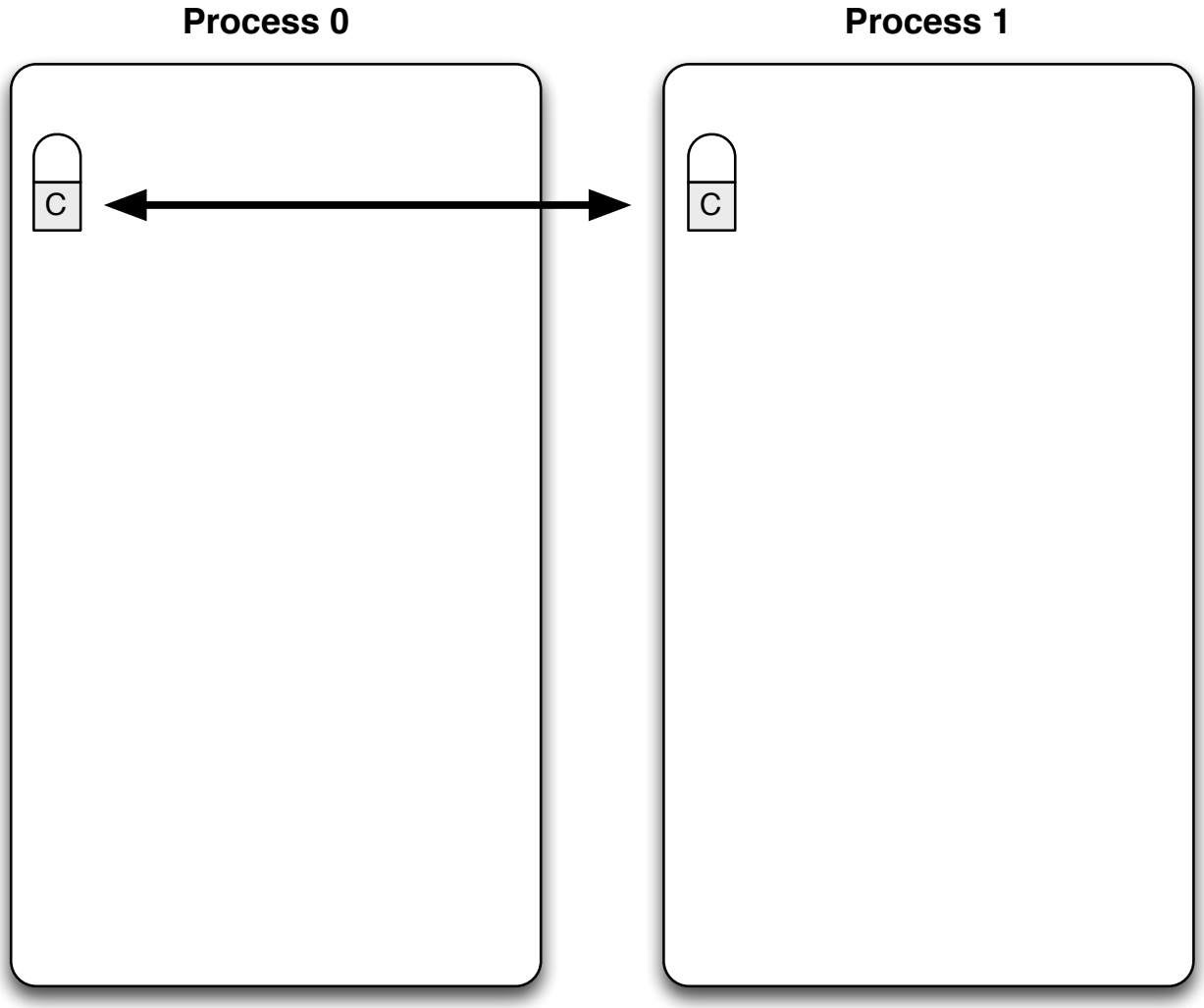

## From the Static to the Dynamic

- Runtime initialization
- Data movement
- Work Migration



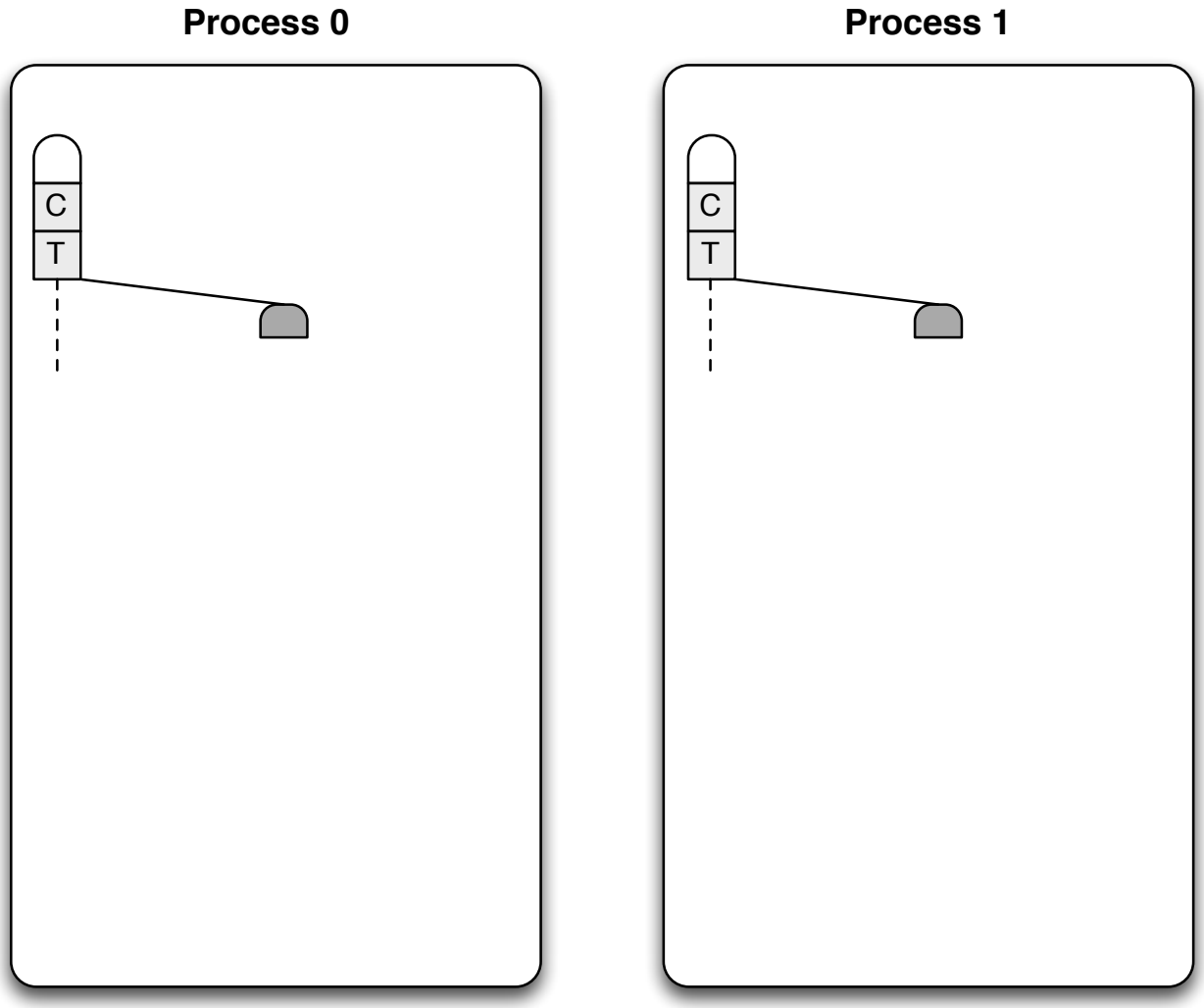

## Parallel Job Launch

- (Skipping the details)
- OS Process == Locale
- SPMD to the runtime



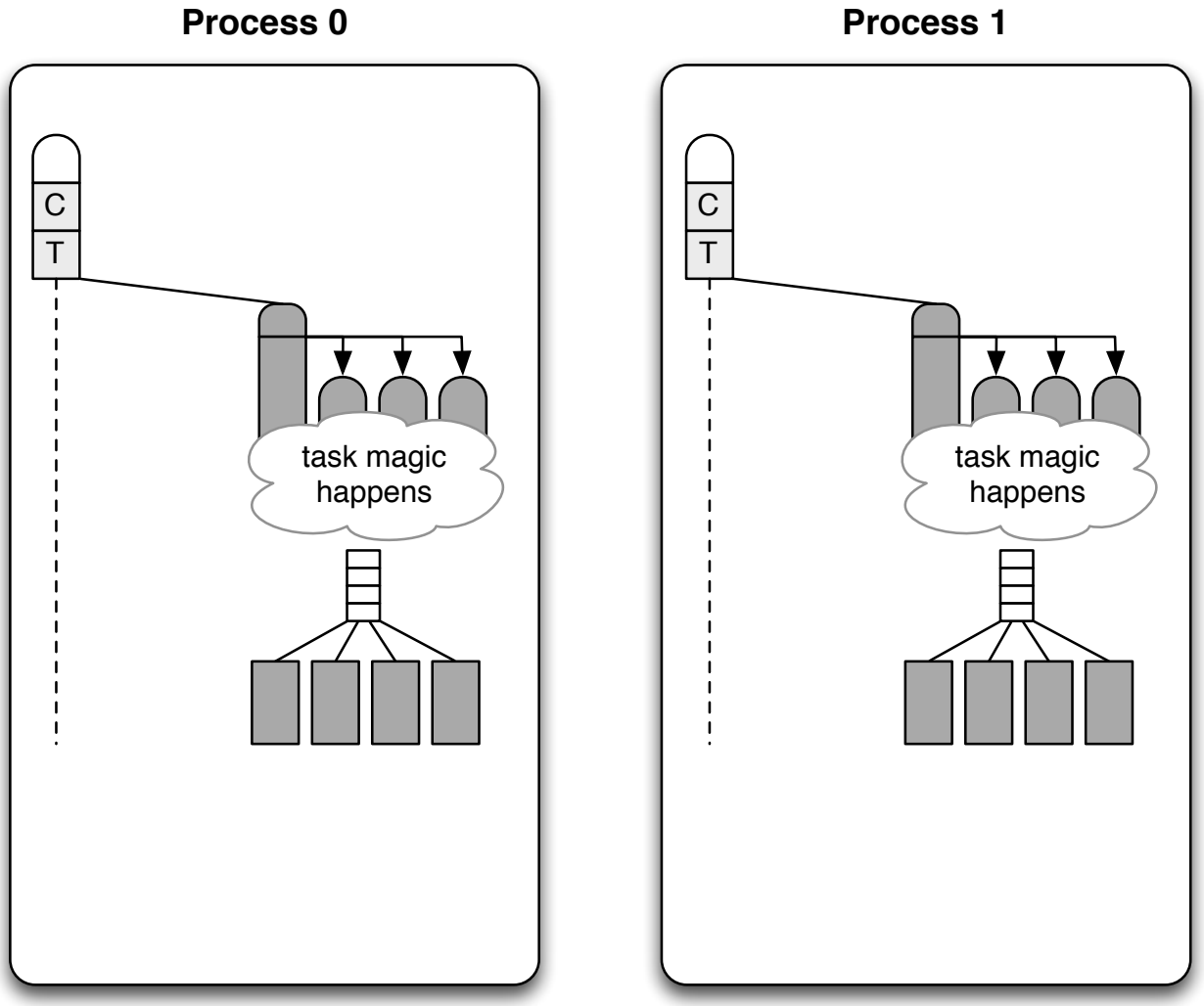

## Comm. Layer Initialization

- chpl_comm_init()
- Action handler registration
- Shared memory segment set up



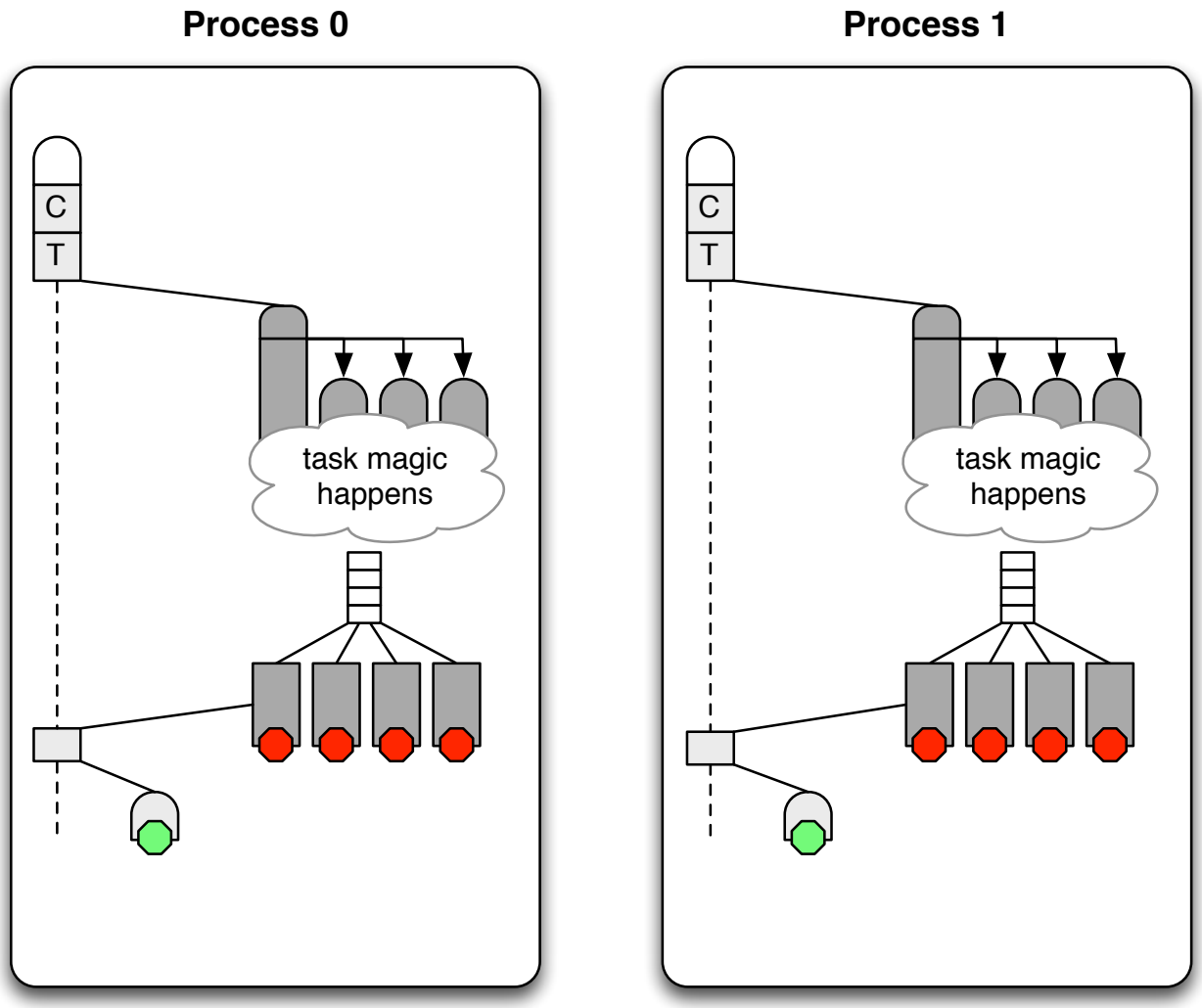

## Task Layer Initialization

- chpl_task_init()
- CHPL_TASKS=qthreads
- Creates a new Pthread for the task layer



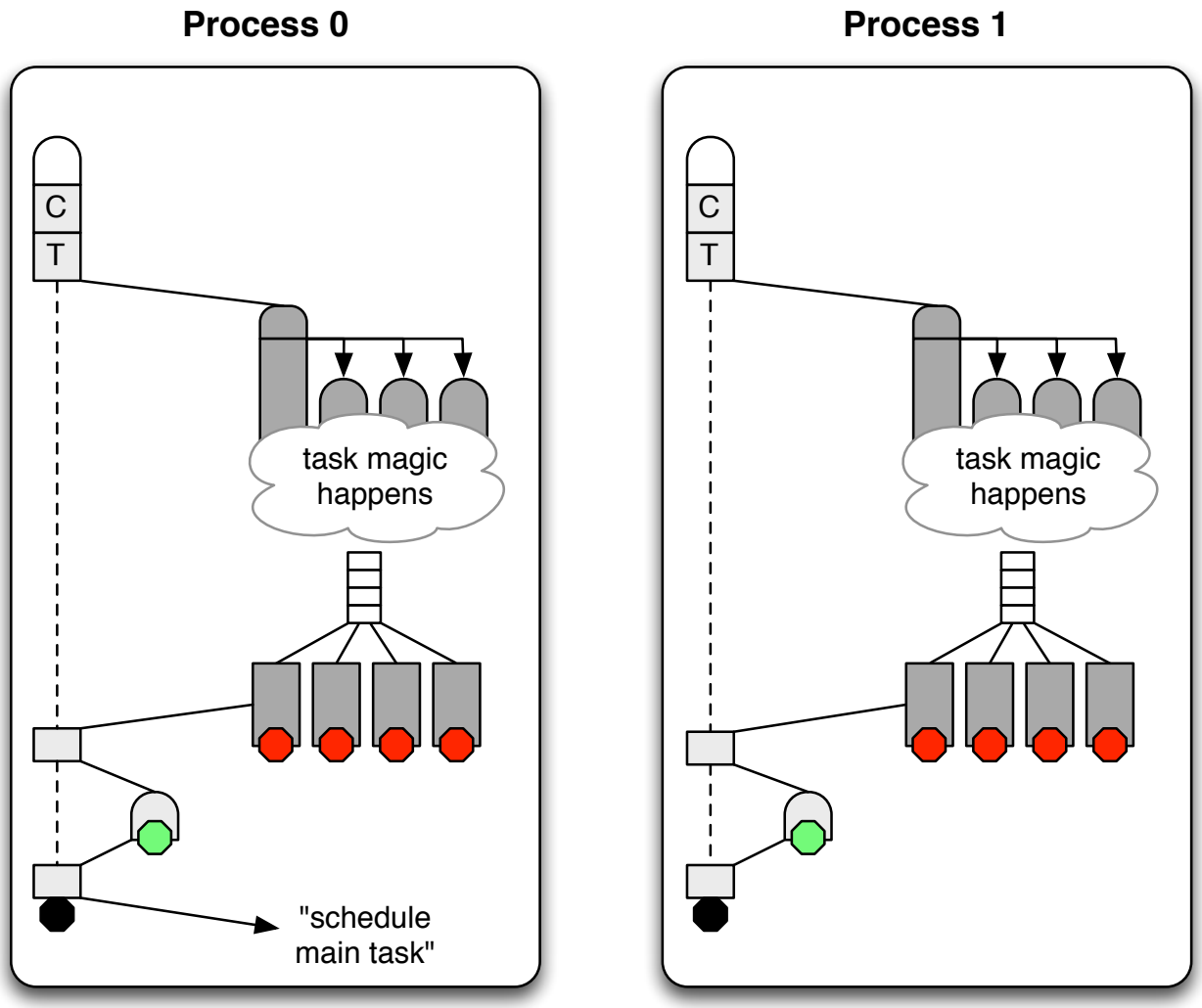

## Task Layer Initialization

- Qthreads is initialized in aux. Pthread context
- Number of worker threads equals number of cores
- Control returns to main Chapel RS thread



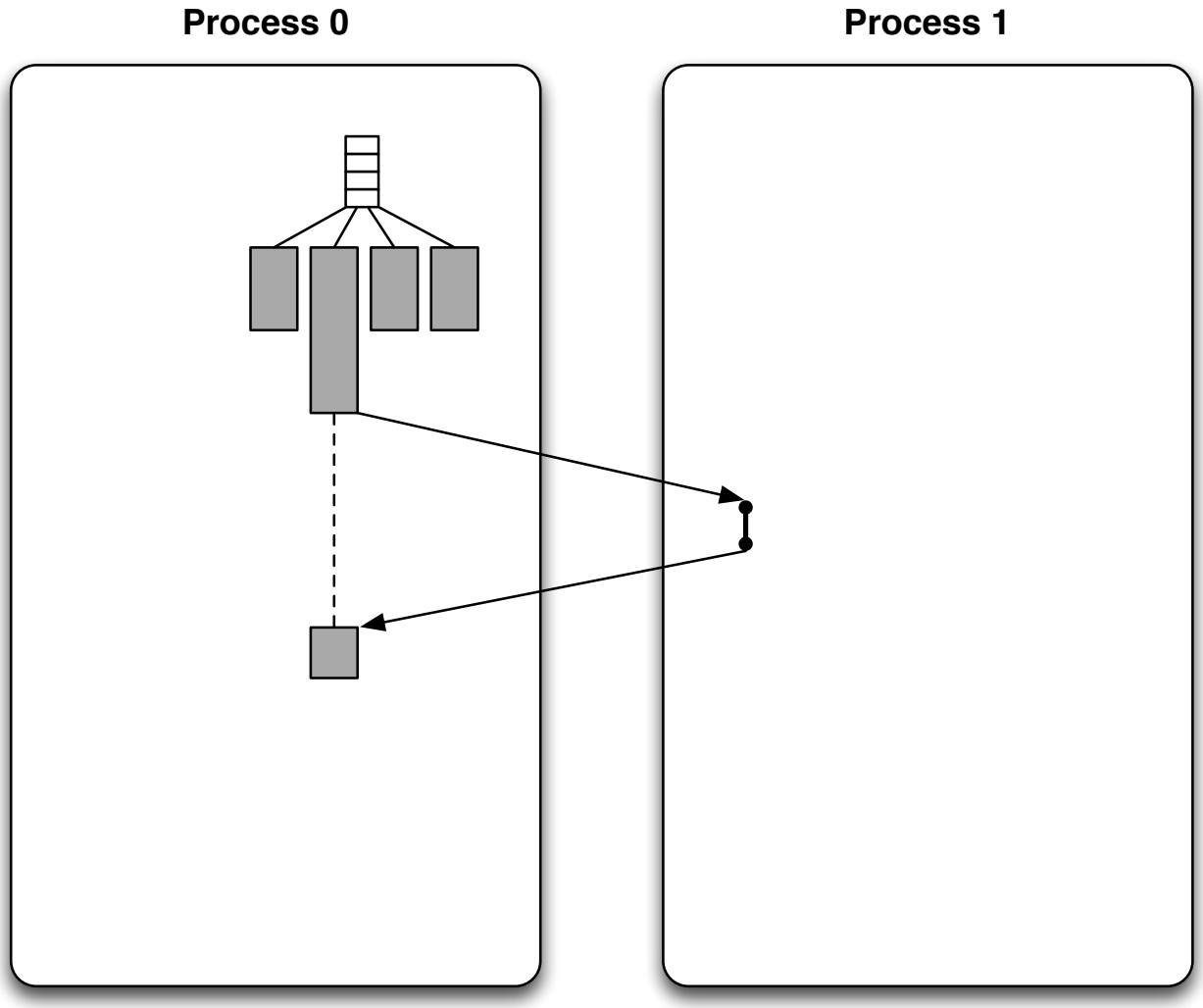

## Progress Engine Start Up

- Another Pthread for a progress engine
- Loop polling GASNet
- chpl_task_yield() converted to OS sched_yield()



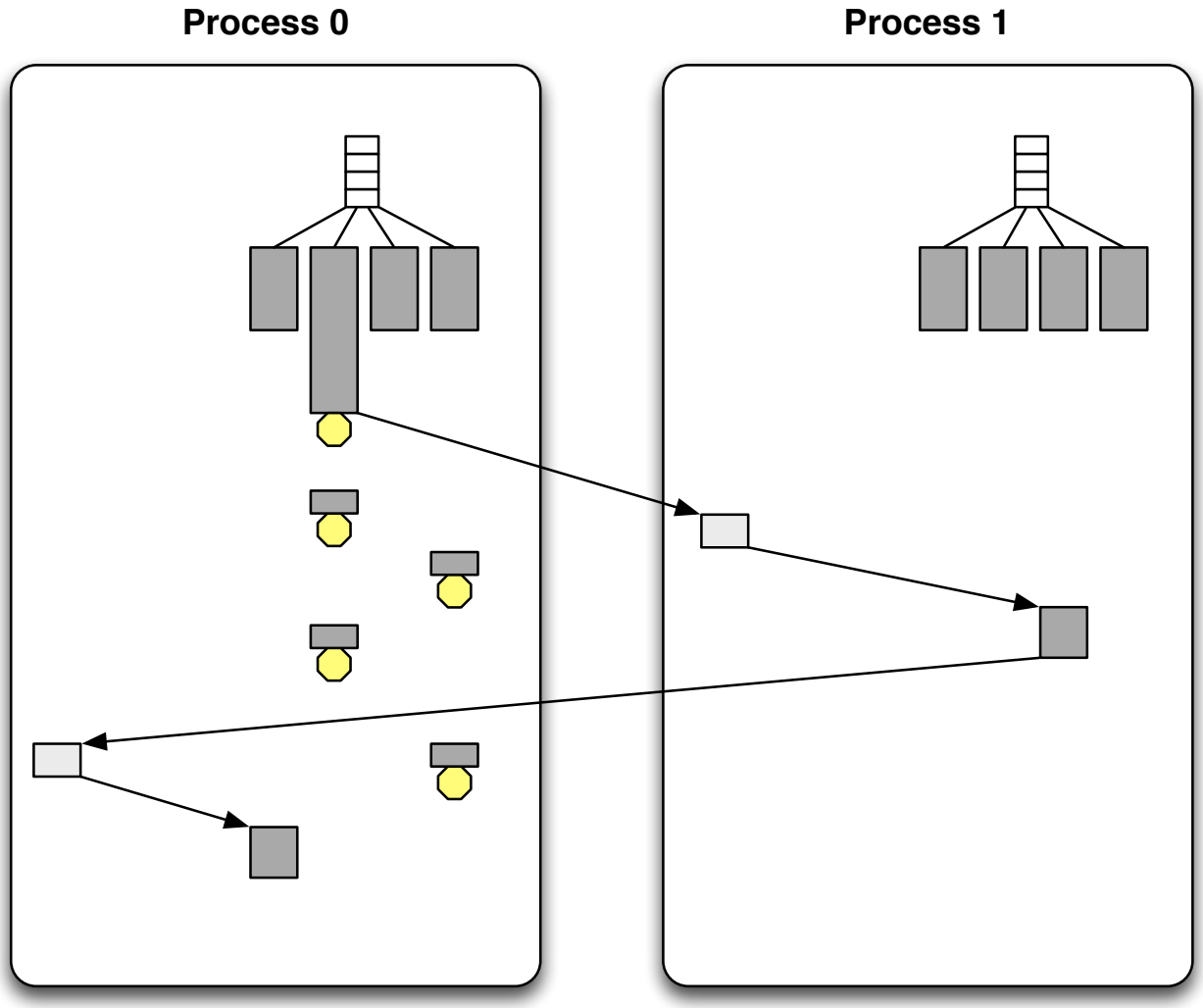

## Application Initiation

- Compiler-generated chpl_main() called to start application code
- Spawned as a task into the tasking layer (from outside)
- Caller “suspends” waiting for that task (really a Pthread mutex block)




## Data Movement

- Put and get operations are implemented in the comm. layer
- Direct mapping to GASNet
- Of note: core **not relinquished** during operation

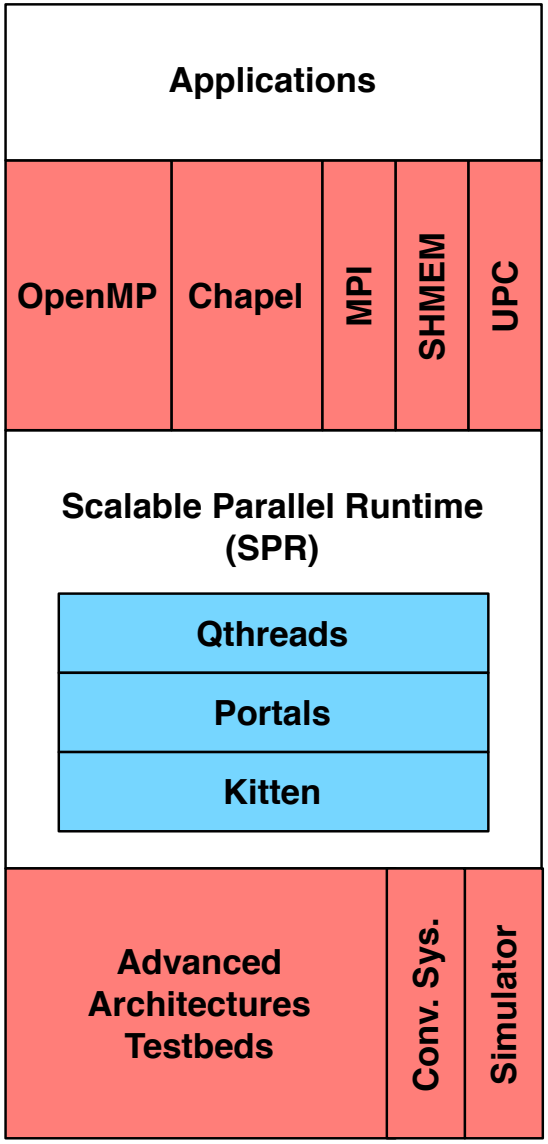



## Work Migration

- 3 types: blocking, non-blocking, and “fast” remote fork
- Calling task loops – polling GASNet for completion and yielding
- Possible scheduler **interference** on the call side

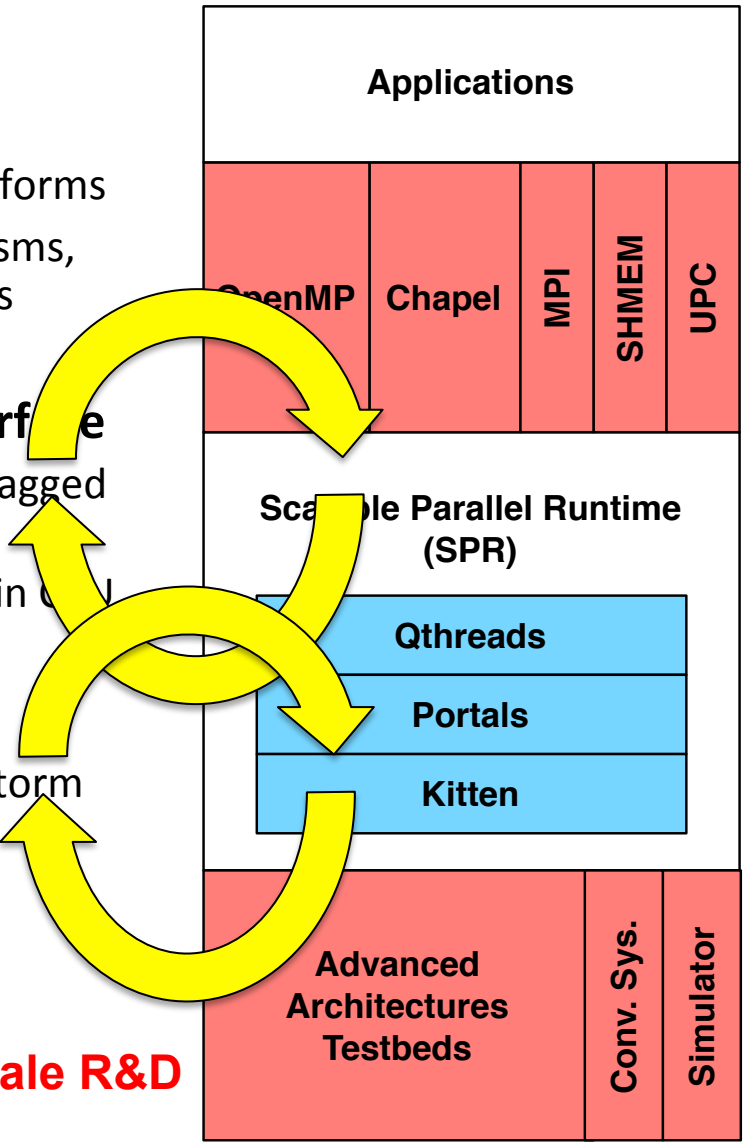
# A Unified Runtime Example

- **Qthreads: Lightweight threading interface**
  - Scalable, lightweight scheduling on NUMA platforms
  - Supports a variety of synchronization mechanisms, including full/empty bits and atomic operations
  - Potential for direct hardware mapping
- **Portals 4: Lightweight communication interface**
  - Semantics for supporting both one-sided and tagged message passing
  - Small set of primitives, allows offload from main CPU
  - Supports direct hardware mapping
- **Kitten: Lightweight OS kernel**
  - Builds on lessons from ASCI Red, Cplant, Red Storm
  - Utilizes scalable parts of Linux environment
  - Primarily supports direct hardware mapping



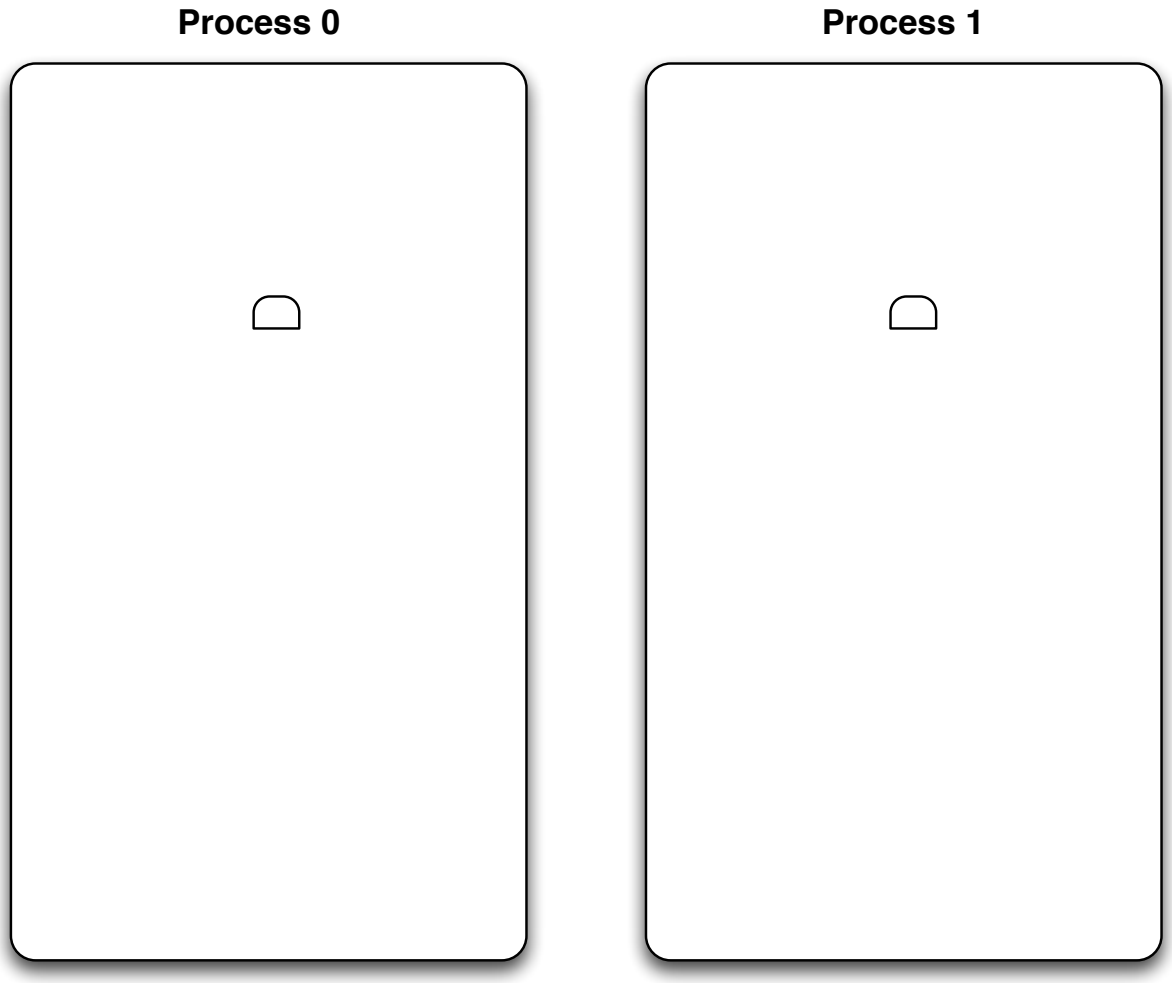
# A Unified Runtime Example

- **Qthreads: Lightweight threading interface**
  - Scalable, lightweight scheduling on NUMA platforms
  - Supports a variety of synchronization mechanisms, including full/empty bits and atomic operations
  - Potential for direct hardware mapping
- **Portals 4: Lightweight communication interface**
  - Semantics for supporting both one-sided and tagged message passing
  - Small set of primitives, allows offload from main CPU
  - Supports direct hardware mapping
- **Kitten: Lightweight OS kernel**
  - Builds on lessons from ASCI Red, Cplant, Red Storm
  - Utilizes scalable parts of Linux environment
  - Primarily supports direct hardware mapping

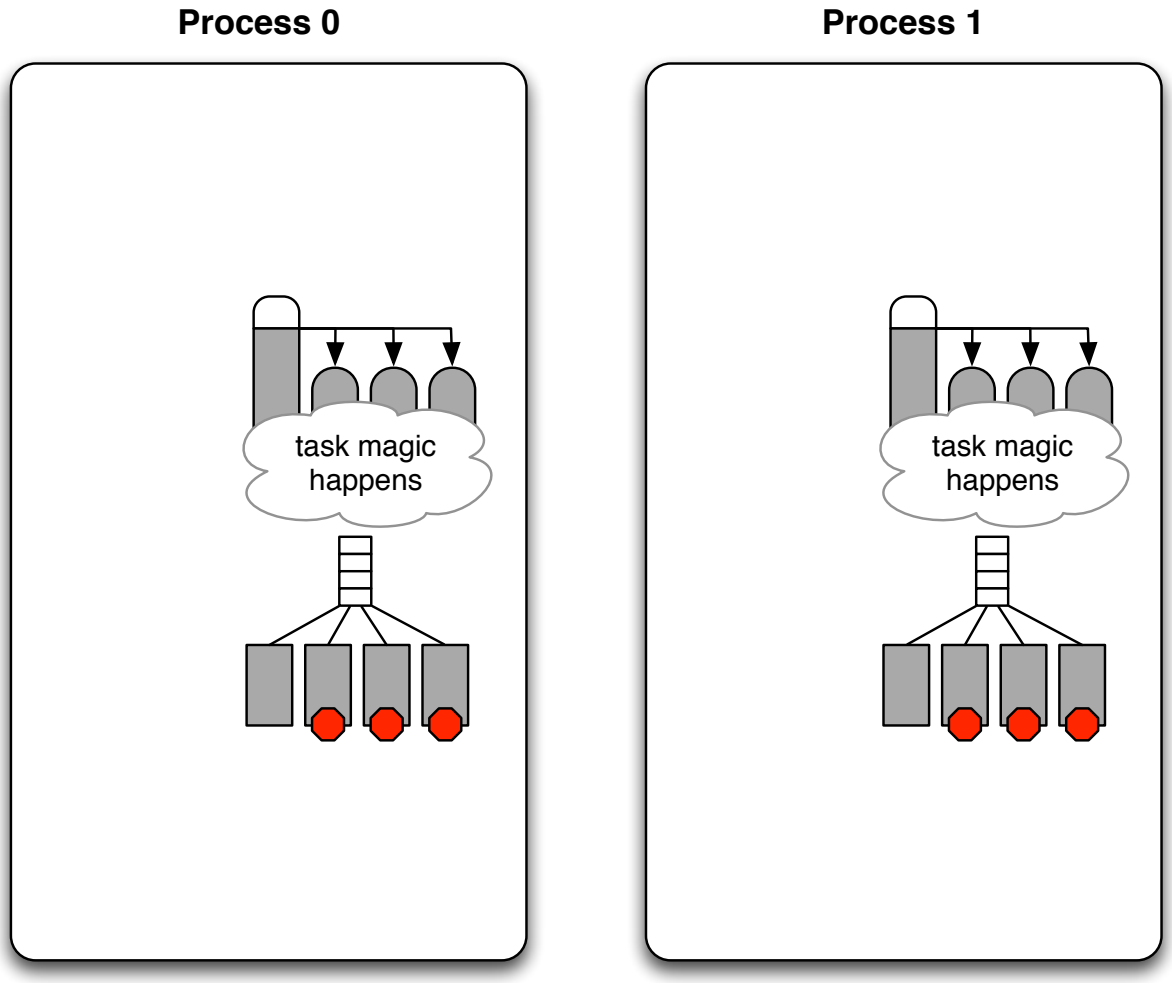



**SPR is the Experimental Platform for Extreme-scale R&D**

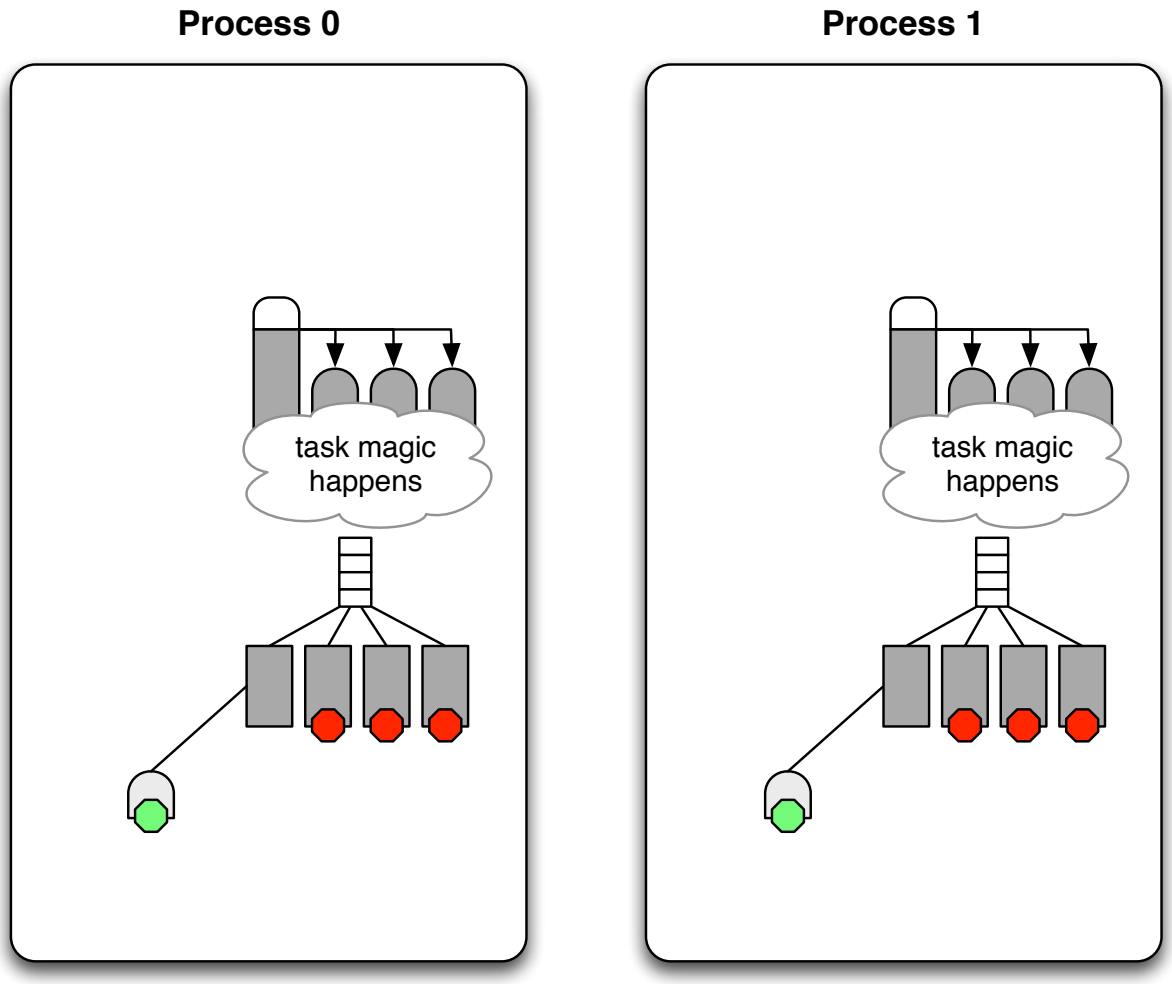
# Task & Network Runtime Init.



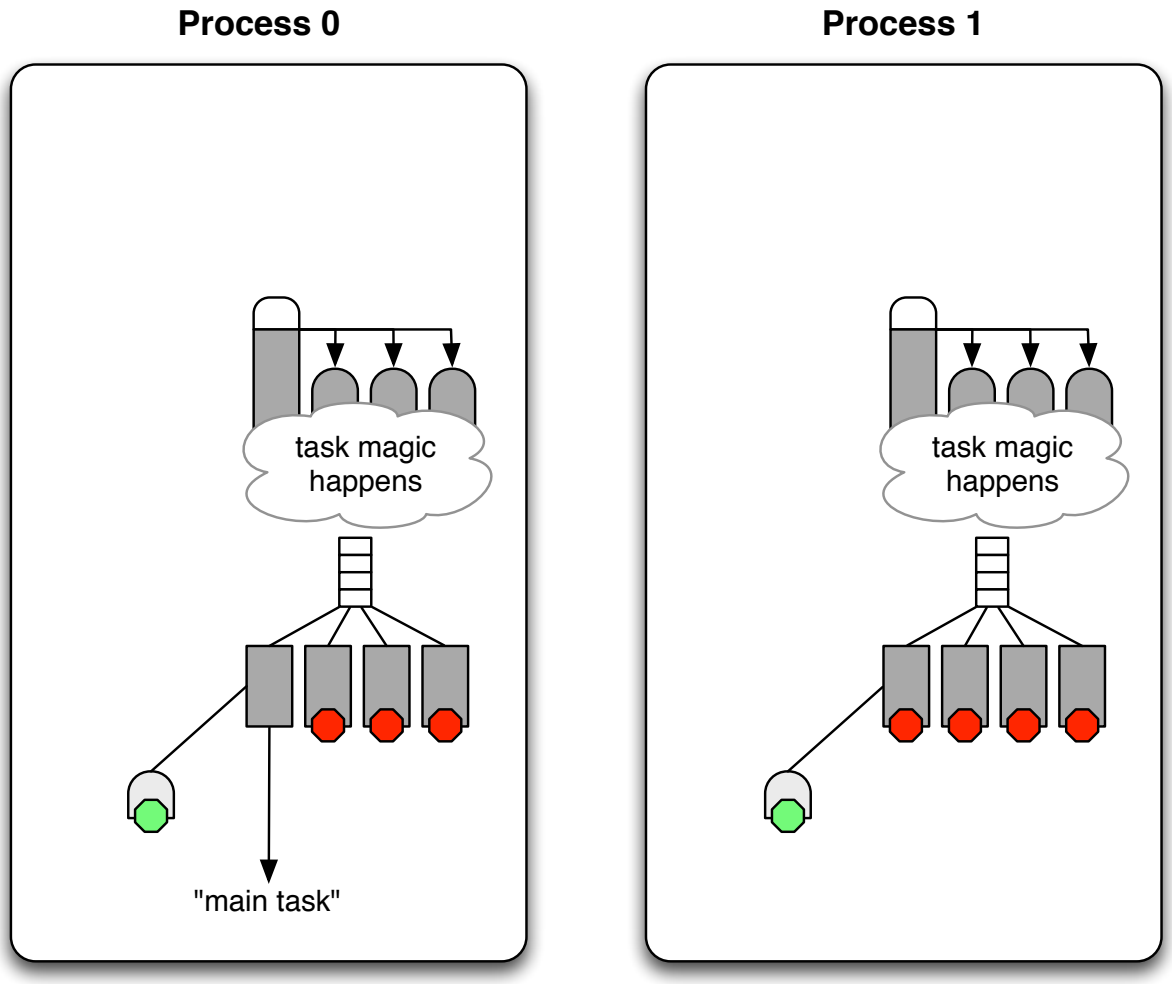
# Task & Network Runtime Init.

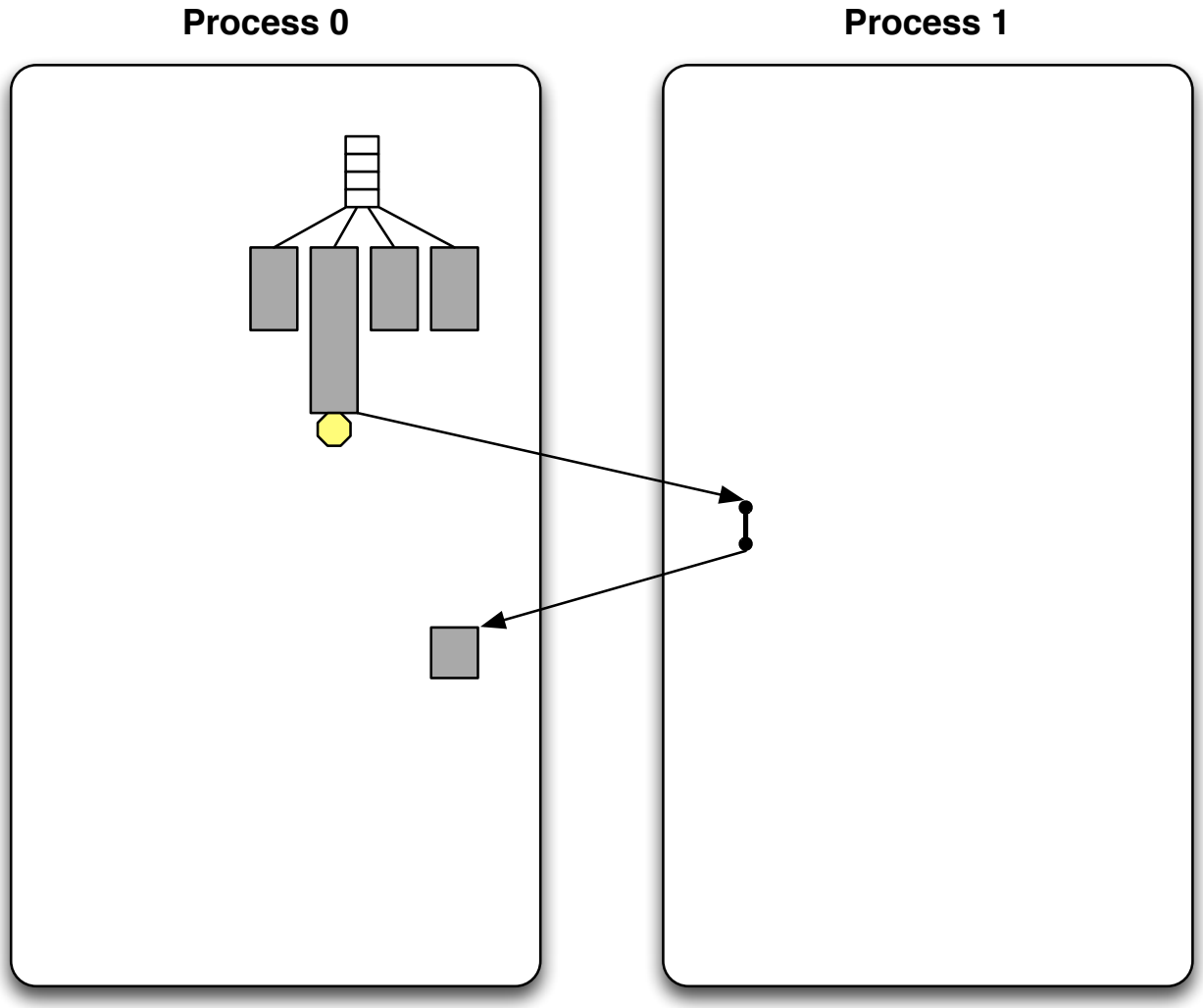


# Progress engine start up



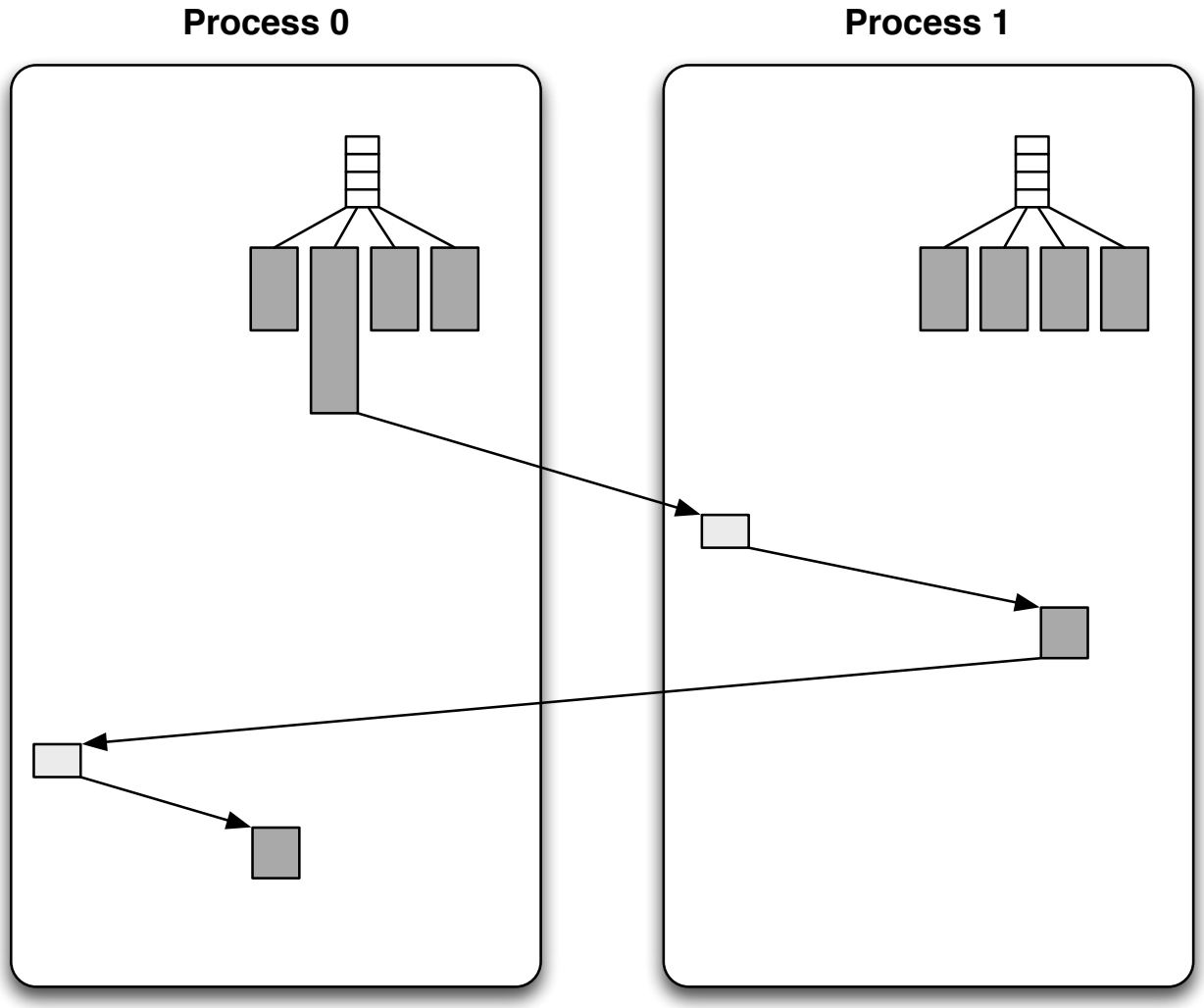
# Application initialization






## Data Movement in the SPR

- Blocking and non-blocking put and get operations
- Calling task suspends, only resumes after completion event
- Progress engine only responsible for FEB operation




Sandia National Laboratories

## Work Migration in the SPR

- Added qthread_fork_remote(..., rank)
- **No synchronization** in the progress engine
- **Remote synchronization** managed through FEB semantics

# Chapel with a Unified Runtime


- Replaced GASNet and Qthreads with SPR
  - Single point for initializing both platforms: spr_init(SPMD,...)
  - spr_unify() used to transition to single thread of control before application starts
  - Most other interface functions are no-ops (e.g., chpl_task_init(), chpl_comm_post_task_init(), chpl_comm_rollcall(), ...)
  - Direct mappings for data movement and work migration

- Cleaner Chapel Runtime Support shim
- Centralized information management

- But just an early **point design**

# Opportunities Moving Forward


- Let third-party implementers worry about
  - Information management
  - Coordinated resource management
  - Integrated local and remote task management
- Reorient Chapel Runtime Support shim interface around unified “locality engine” (CHPL_LE=?)
  - Remove runtime silos
  - Focus on communicating information about data movement and work migration to unified runtime layer
  - Open up runtime ecosystem to the increasing assortment of unified runtimes: HPX, GMT, Grappa, etc.
- Start a runtime-centric working group to coordinate efforts between compiler writers and RS implementers