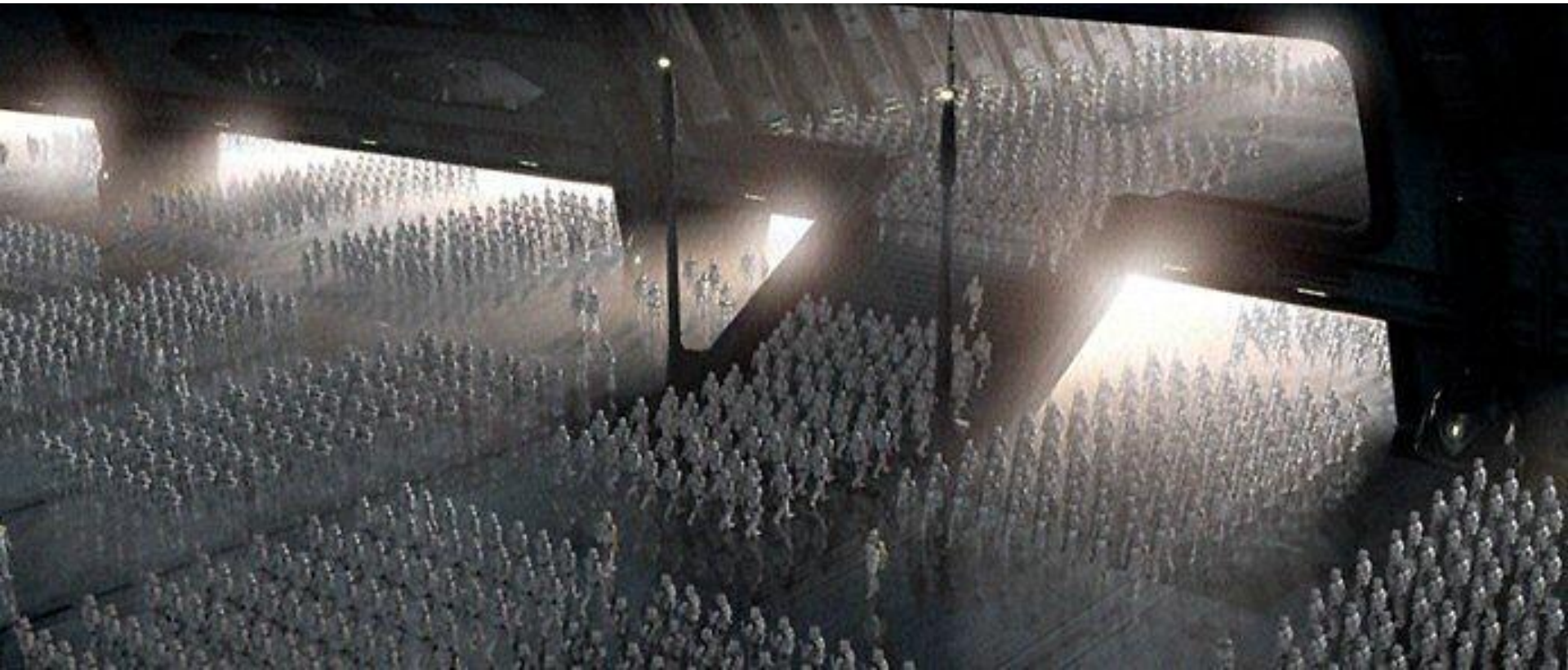
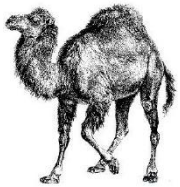
Threading and Perl – avoiding insanity and managing the clones effectively

YAPC::NA 2014 - David Bury



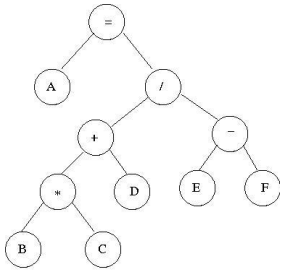
About Me?

- Live in Monterey, CA (moving to San Diego, CA)
- About 10 years in software



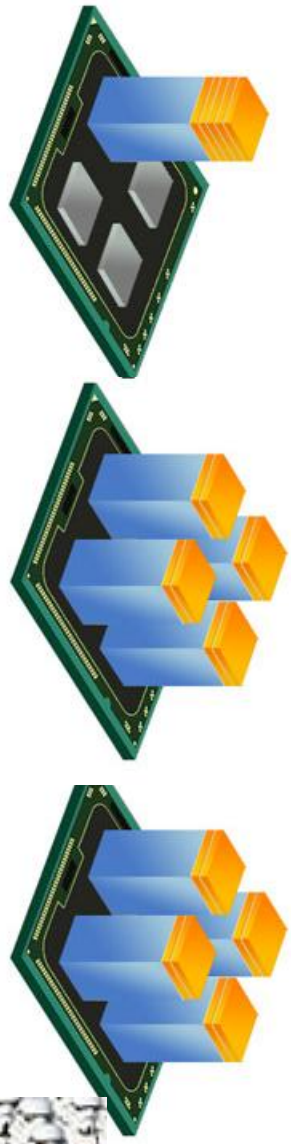
  - Recently back/forth Perl and Java.
  - Lots of focus on data back ends and sets.

  - Enjoy playing with a parser or two
  - Moving to mess around with big data problems.



- First YAPC::NA presentation
- dsbike@gmail.com for contact.

Why Use Threads?



- A normal, single threaded, program is unusable
  - A client UI – Can't do graphics and work and be usable.
- Performance and use of CPU matters
  - Normal program uses one core out of many.
  - Matters with 4, 16, 32 cores. Cores increasing.
- High latency Operations
  - Cost is in wait time, sequential waits are slow.
  - Parallel waiting faster and better design.

The Catch

Normal Code + Threads = Pandemonium

All Potential Normal linear code problems

Non-deterministic behavior

Simultaneous story lines

Story line interactions

linear code problems

X

Randomness

X

Story lines Emergent Behavior

Controlling Emergent Behavior

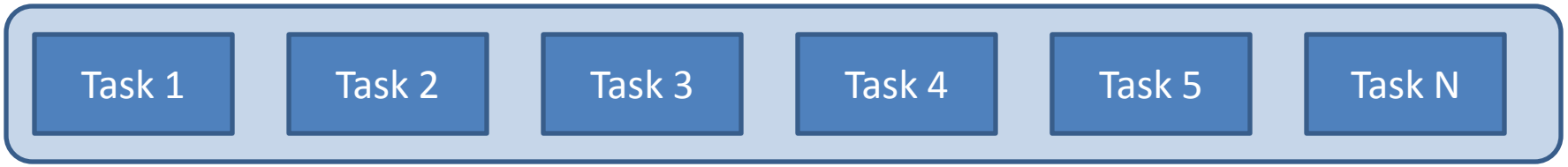
Design of runtime interactions and multithreaded components matter. Use well defined design patterns.

Examples

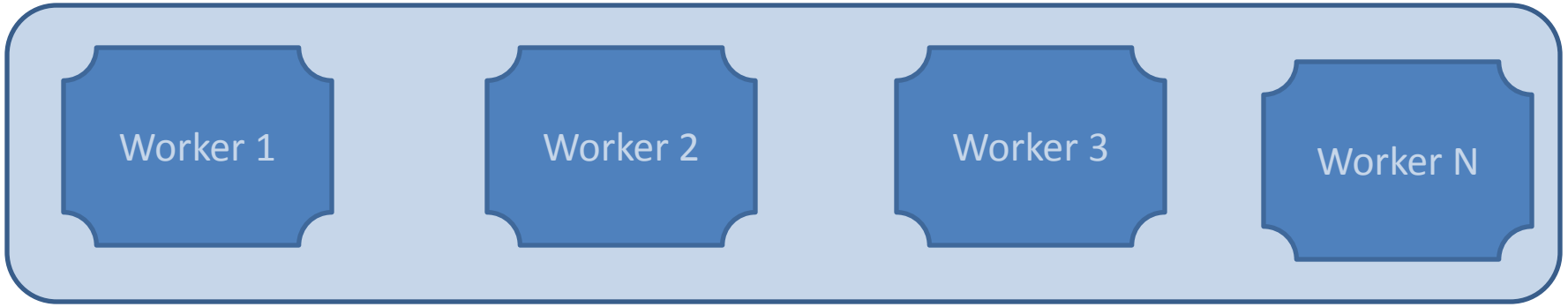
Pattern	Summary
Barrier	No thread progress at certain points
Double-checked locking	Use a lock with a pre-lock to lock better
Reactor	Event based concurrent execution
Reader-writer lock	Alternation of read/write locking
Thread Pool	Queue based distribution of work

The Thread Pool

Queue – Contains ordered list of Tasks to complete:

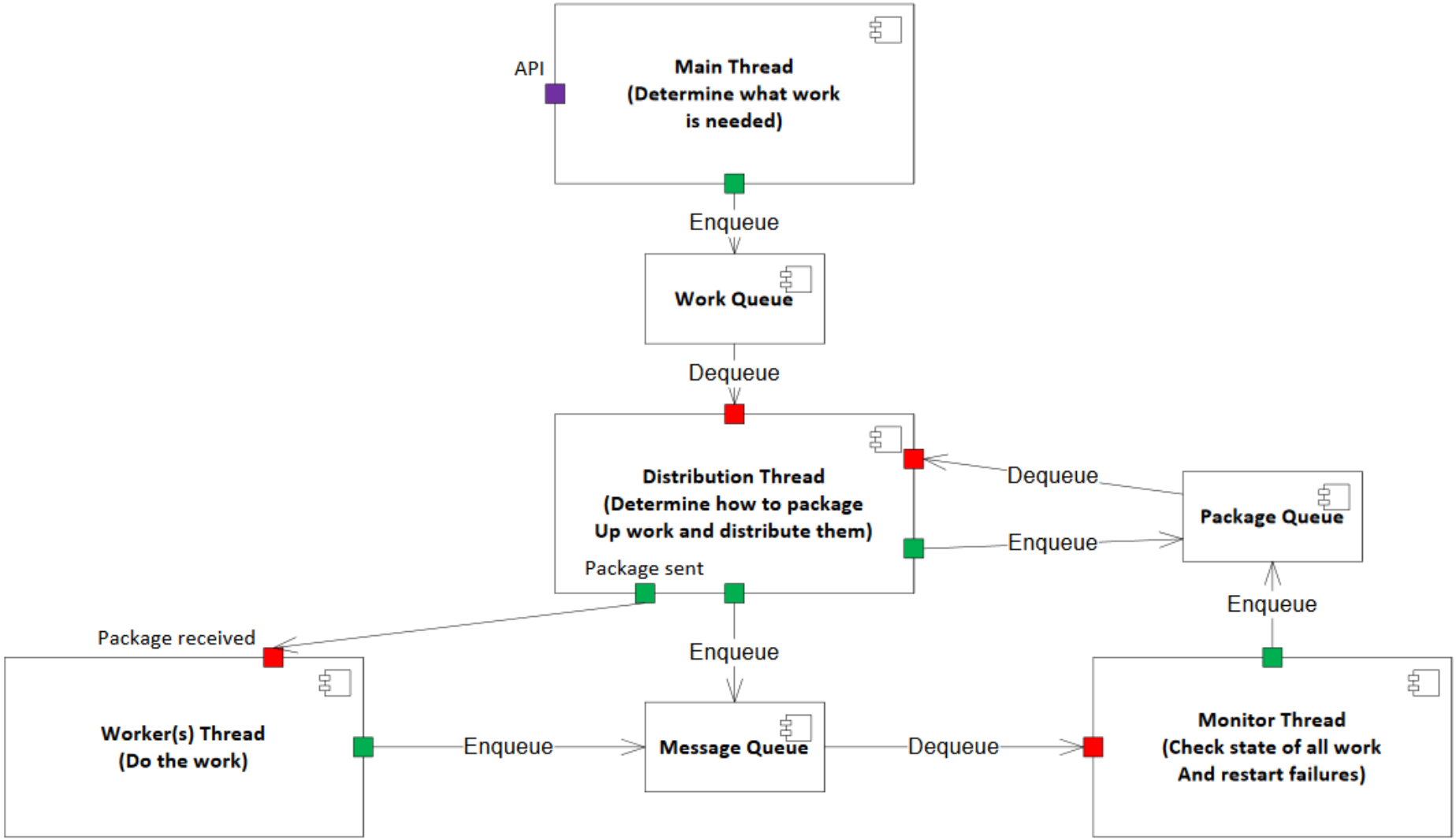


Pool – Contains N Threads that process concurrently:



Like having one line of customers at a grocery store for checkout; each person goes to the first free N cashiers. Customer = task, cashiers = pool.

Thread Pool Example



Perl Threads 101

- “use threads”
- “threads->create(‘function_name’, arguments)”
- Runs in a new thread function_name with your arguments.
- Argument Pitfalls:
  - Not completely normal function call
  - More than one level in hash or array = missing data in thread.
  - Complex data needs threads::shared (next slide)

threads::shared 101

- "use threads::shared", only after "use threads"
- Basic use
  - "my $var :shared"
  - "{ lock($var); # read/write}"
- Limitations
  - Only works on scalars, arrays, hashes, + refs.
  - No glob, code ref support.
  - Caution on array of array, hash of hash
  - "share_clone($data)" helps but copies **values not references**.

Communication between Threads

- Basic
  - Locks with threads::shared – tracking/coordination variable(s) of simple state.
- Semaphores
  - The thread safe count – 0/1 like a basic locking, counter for resource sharing.
- Other
  - Thread::Queue – Queues that are thread safe.

Thread::Queue

- Works very nicely with Thread Pool pattern.
- Have multiple threads write to this: state/work
- Have one responsible thread read/distribute.
- “my $queue = Thread::Queue->new()”
- Pass queue object freely to threads (as param)
- Write to queue: “$queue->enqueue($data)”
- Read off queue
  - “$queue->dequeue_timed($seconds, $number_wanted)”

Testing with Threads

Mocking test classes usually depend on manipulating object in memory

- Fails in most thread testing
- Possible only on some object/function structures

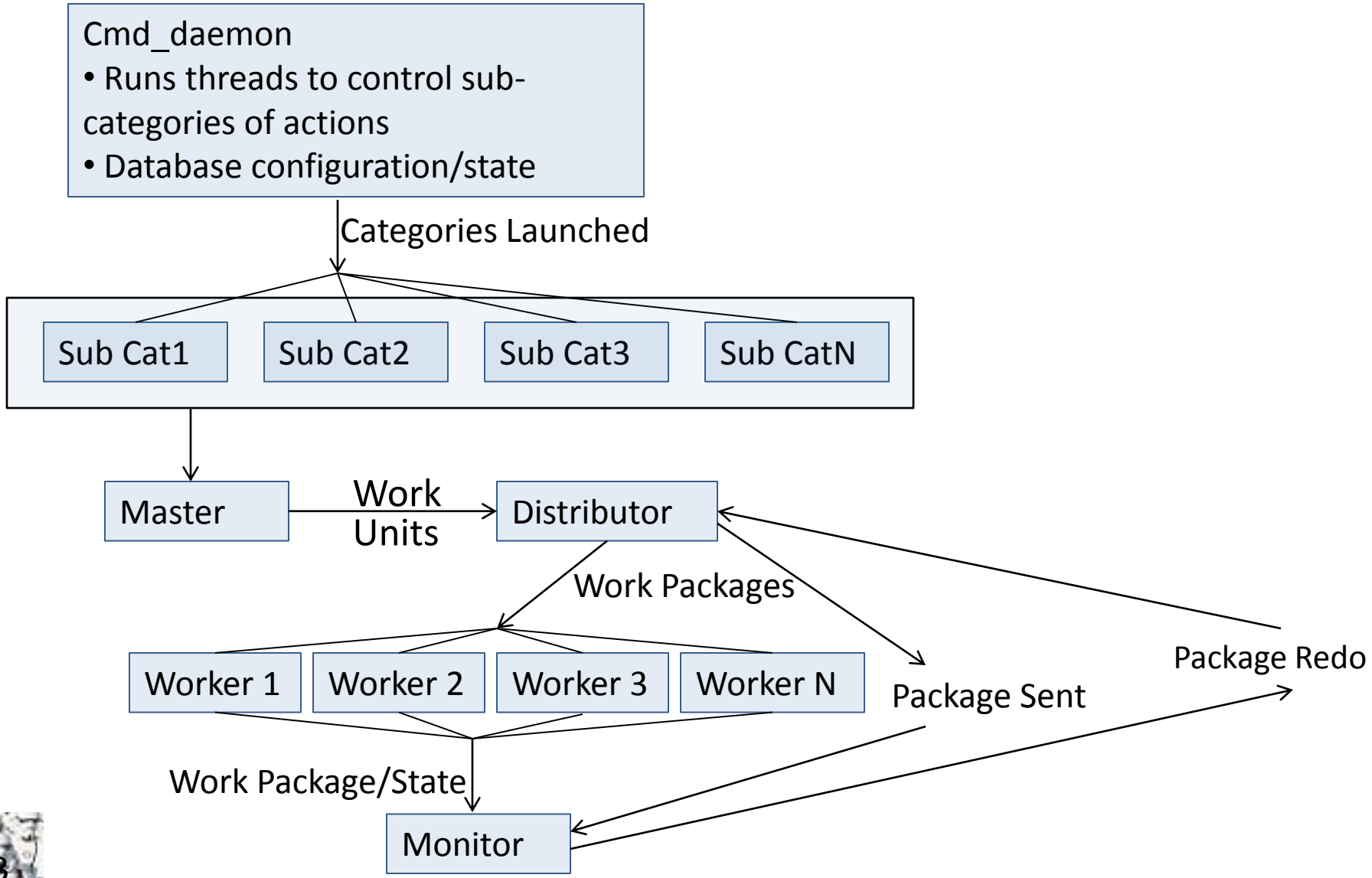
Solution: Mock Statically – physically modify libs and files for testing.

- Use test area of overriding libraries
- Create mock of in memory object
- Write object file to test area; Reload libraries
- Run your test.

Advanced

- Use file based variable set/get with static mocking
- Handles changes to mock object behavior for variety of test scenarios.

Real life Example



Conclusion

- Use when needed only, non-trivial
- Runtime logic especially critical to understand
- Design pattern helpful for design or starting point
- Simplicity in data sharing helps system management
- Understand thread API and limitations with threads
- Unit Test, Integration Test, Thread Test

Thank You

Further Question/Comments: dsbike@gmail.com