Applied Parallel Computing With Python

Ian Ozsvald
ian@morconsulting.com
www.morconsulting.com

Minesh B. Amin
mamin@mbasciences.com
www.mbasciences.com

PyCON 2013
Santa Clara, CA
March 14, 2013

Bio (Minesh)

’97-’05 Developed Commercial $\left\{\begin{array}{c}\text{Serial}\\ \text{Parallel}\end{array}\right\}$ Enterprise Software *still* used by Hardware Engineers

’06- MBA Sciences, Inc

Self-funded Engineer/CTO/Founder $\rightarrow$ SPM.Python + Consulting Services

$\downarrow$

Disruptive Technology

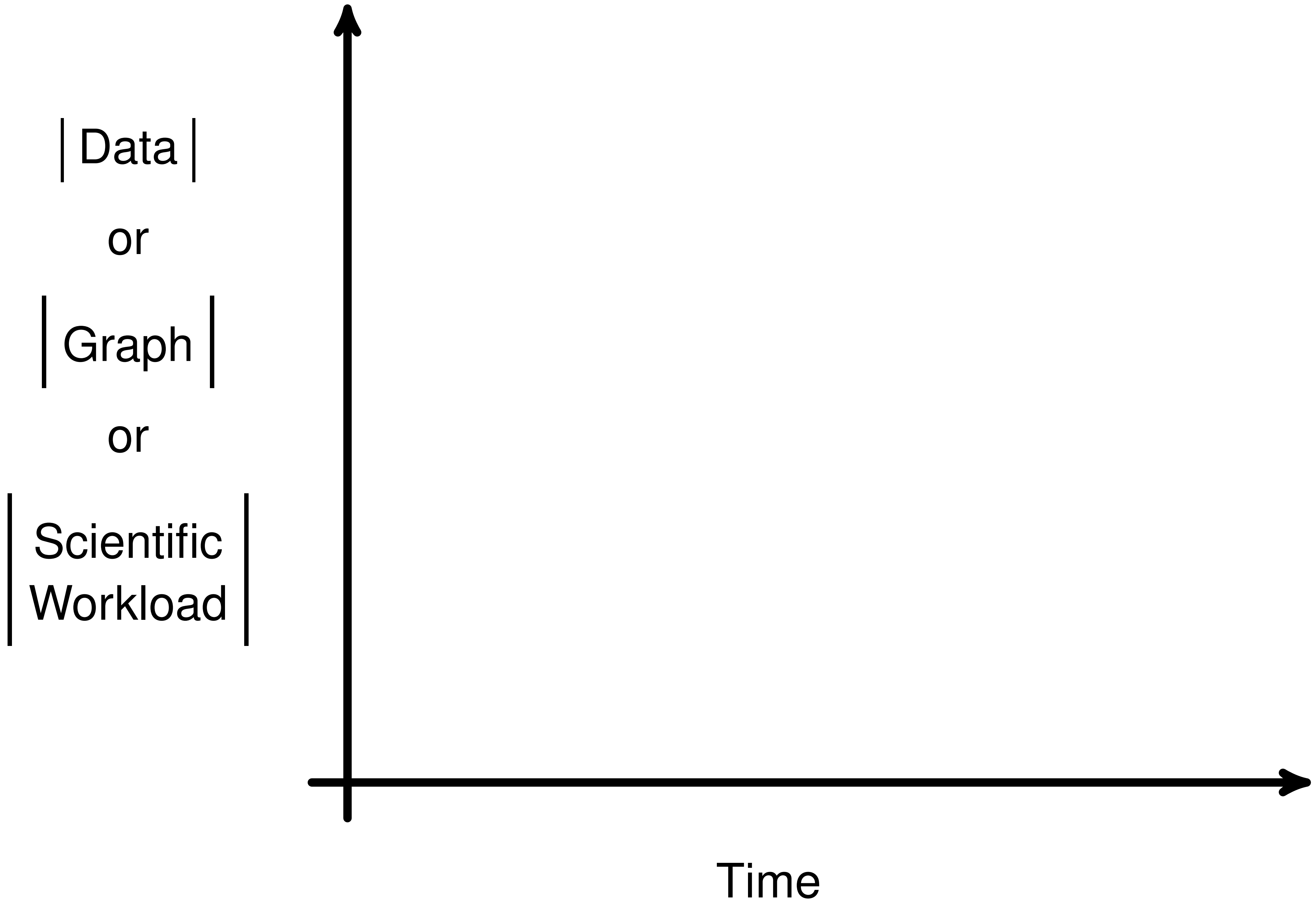
Supercomputing Conference 2010

GTC 2012

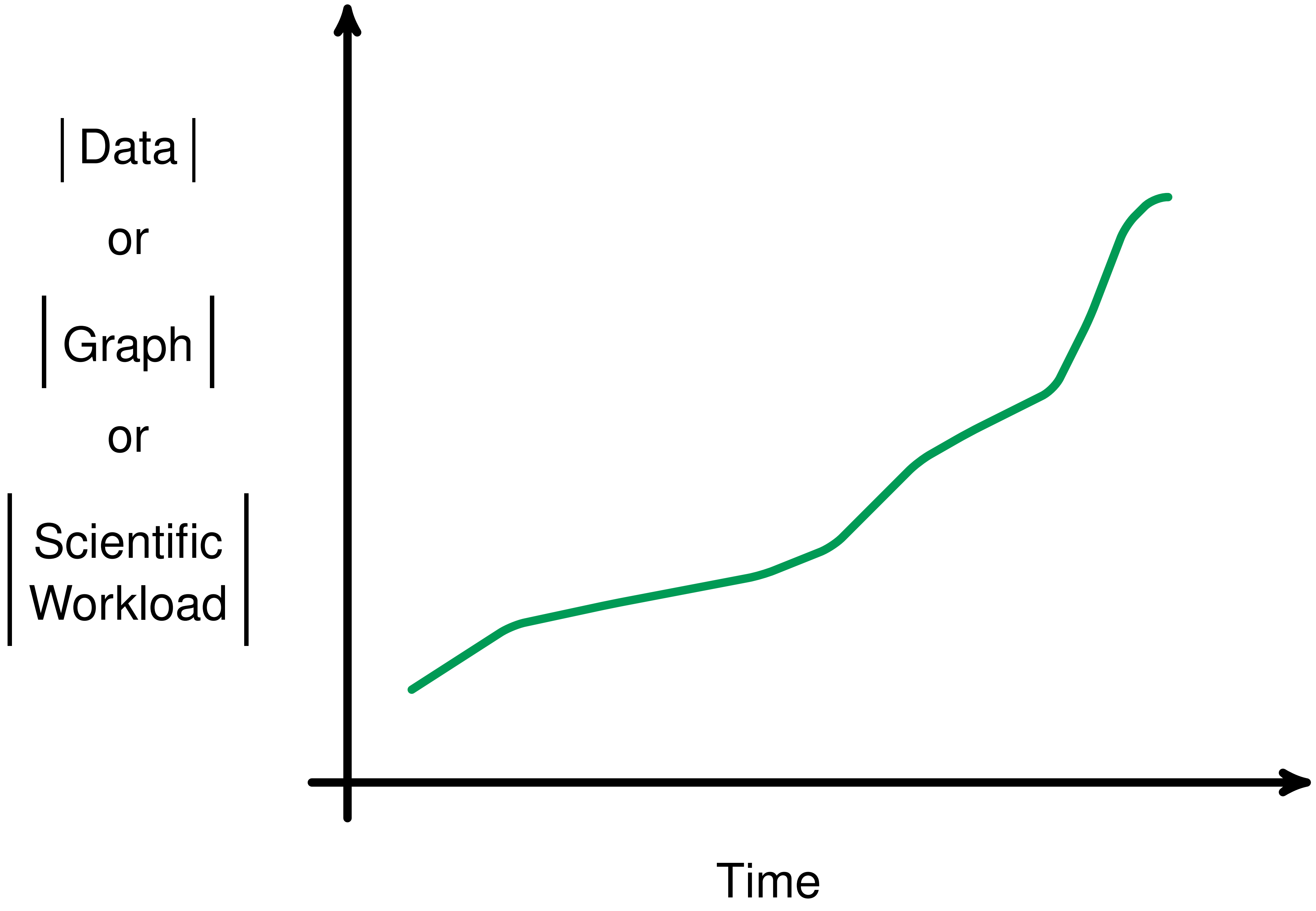
Footnote on the term ...

"Exploiting Parallelism"

{Robust / Fault tolerant} Parallelism: Why?



{Robust / Fault tolerant} Parallelism: Why?



$\left\{\begin{array}{c}\textbf{Robust}\\\textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take I)?

Serial Module

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s       " \
                         "%(id)s        " \
                         "%(dotConfig)s " \
                         "%(dotTgz)s    " \
                         "%(rankD)s     " % dict(bin       =           bin,
                                                 id        = remoteArgs.id,
                                                 dotConfig = remoteArgs.dotConfig,
                                                 dotTgz    = remoteArgs.dotTgz,
                                                 rankD     = rankD,
                                                 ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

{Robust / Fault tolerant} Parallelism: How (Take I)?

Serial Module

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
             (cmd     = "%(bin)s       " \
                        "%(id)s        " \
                        "%(dotConfig)s " \
                        "%(dotTgz)s    " \
                        "%(rankD)s     " % dict(bin       =           bin,
                                                id        = remoteArgs.id,
                                                dotConfig = remoteArgs.dotConfig,
                                                dotTgz    = remoteArgs.dotTgz,
                                                rankD     = rankD,
                                                ),
              timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple Invocations


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
             (cmd     = "%(bin)s       " \
                        "%(id)s        " \
                        "%(dotConfig)s " \
                        "%(dotTgz)s    " \
                        "%(rankD)s     " % dict(bin       =           bin,
                                                id        = remoteArgs.id,
                                                dotConfig = remoteArgs.dotConfig,
                                                dotTgz    = remoteArgs.dotTgz,
                                                rankD     = rankD,
                                                ),
              timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

{Robust, Fault tolerant} Parallelism: How (Take I)?

Serial Module

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =             bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple Invocations


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =             bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```


→ Parallel Module


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =             bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

{Robust / Fault tolerant} Parallelism: How (Take I)?

Serial Module

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s  " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =           bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple Invocations


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s  " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =           bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```


→ Parallel Module

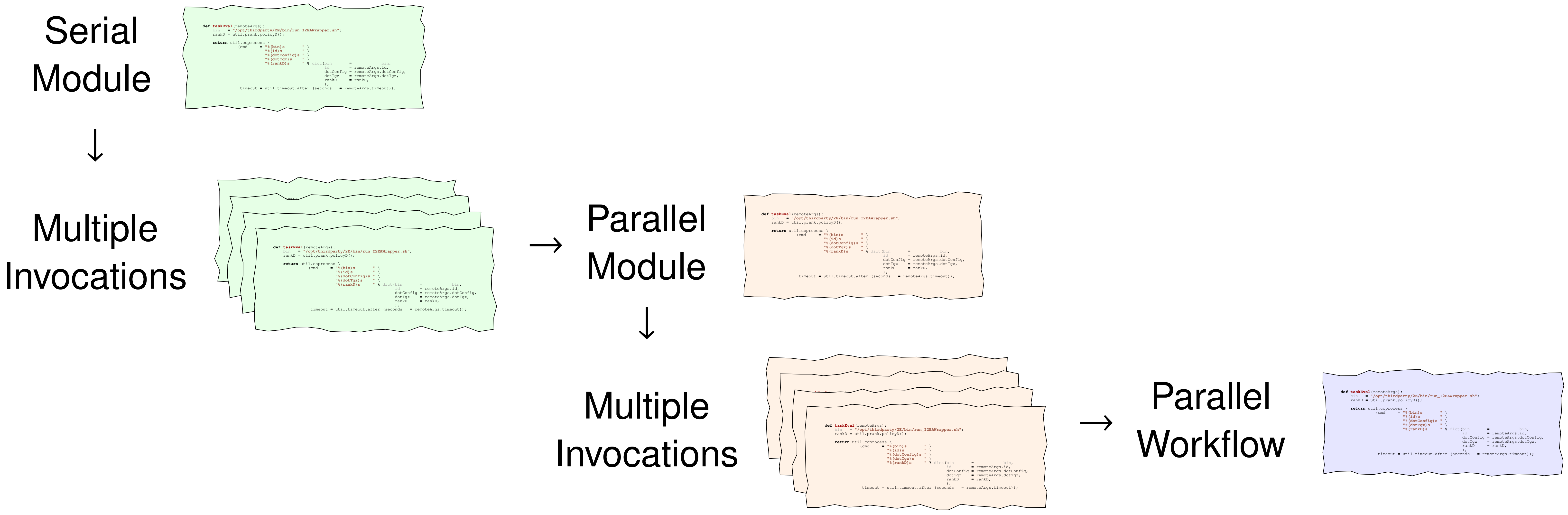

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s  " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =           bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```


↓

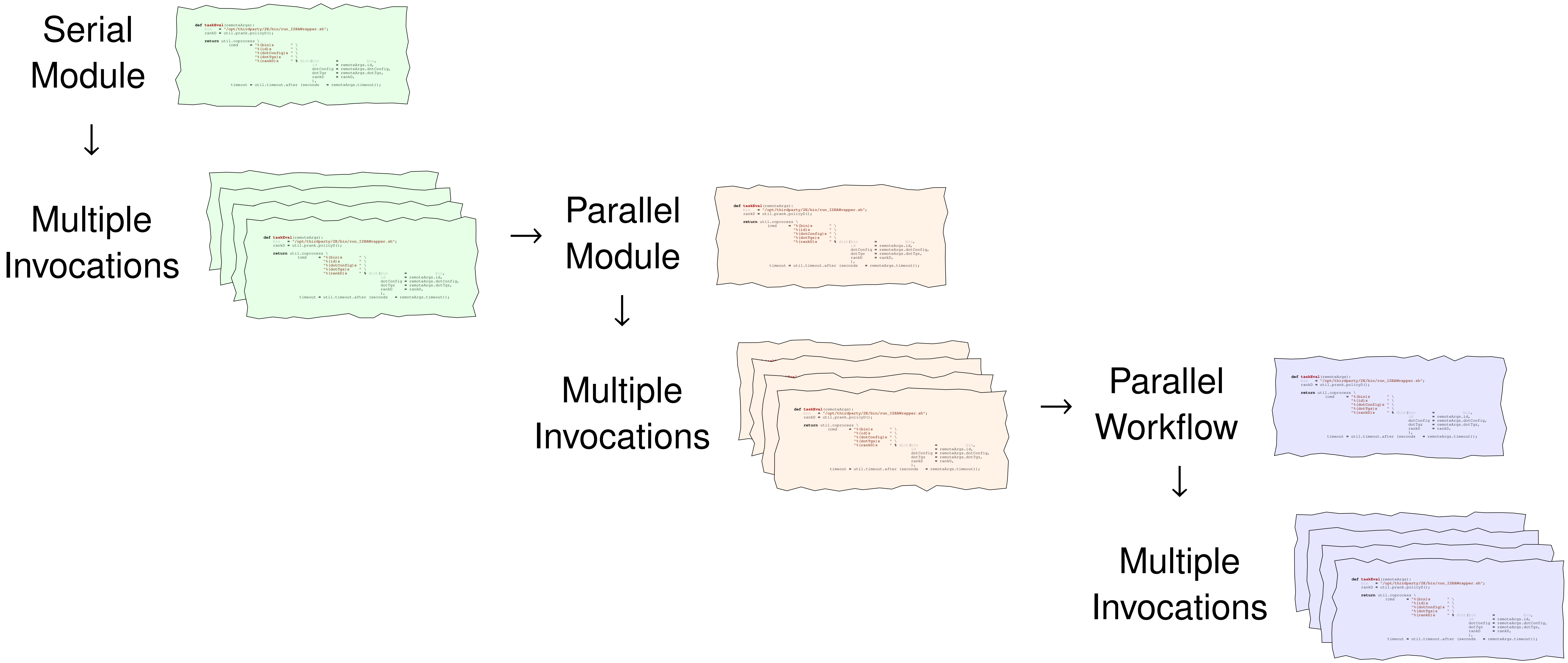
Multiple Invocations


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
              (cmd     = "%(bin)s        " \
                         "%(id)s         " \
                         "%(dotConfig)s  " \
                         "%(dotTgz)s     " \
                         "%(rankD)s      " % dict(bin       =           bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
               timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

$\left\{ \begin{array}{c} \textbf{Robust} \\ \textbf{Fault tolerant} \end{array} \right\}$ Parallelism: How (Take I)?



{Robust / Fault tolerant} Parallelism: How (Take I)?



$\left\{\begin{array}{c}\textbf{Robust}\\\textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take II)?

A language determines the concepts we can think of

- Benjamin Worf

$\left\{\begin{array}{c}\textbf{Robust}\\\textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take II)?

$$\text{A} \left\{\begin{array}{c}\text{language}\\ \\ \\ \end{array}\right\} \text{determines the concepts we can think of}$$

$\left\{\begin{array}{c}\textbf{Robust}\\ \textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take II)?

$$\text{A} \left\{\begin{array}{c}\text{language}\\ \text{runtime env}\\ \\ \\ \end{array}\right\} \text{determines the concepts we can think of}$$

$\left\{\begin{array}{c}\textbf{Robust}\\\textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take II)?

$$\text{A} \left\{\begin{array}{c}\text{language}\\\text{runtime env}\\\text{framework}\\ \\ \end{array}\right\} \text{determines the concepts we can think of}$$

$\left\{\begin{array}{c}\textbf{Robust}\\ \textbf{Fault tolerant}\end{array}\right\}$ Parallelism: How (Take II)?

$$\text{A}\ \left\{\begin{array}{c}\text{language}\\ \text{runtime env}\\ \text{framework}\\ \text{library}\end{array}\right\}\ \text{determines the concepts we can think of}$$

Preamble: "The Big Picture"

Question: Is exploiting parallelism $\left\{ \begin{array}{c} \text{easy} \\ \text{hard} \end{array} \right\}$?

Preamble: "The Big Picture"

What makes

Question: Is exploiting parallelism $\left\{ \begin{array}{c} \cancel{\text{easy}} \\ \text{hard} \end{array} \right\}$?

Preamble: "The Big Picture"

What makes

Question: ~~Is~~ exploiting parallelism $\left\{\begin{array}{c}\cancel{\text{easy}} \\ \text{hard}\end{array}\right\}$?

Copyright 1994, The UNIX-HATERS Handbook

Supposition: The gap between developer’s intent and API of PET
(parallel enabling technologies) ...

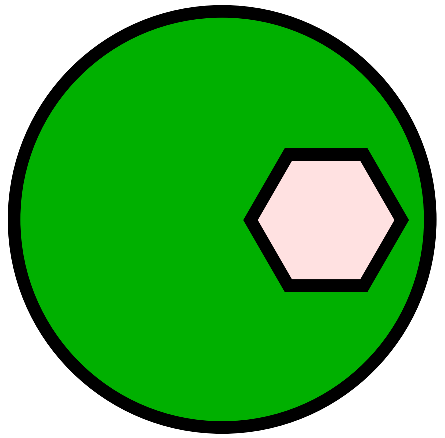
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

OpenMPI OpenMP
CUDA OpenGL

- Maximum flexibility
- Maximum headaches
- Must implement fault tolerance



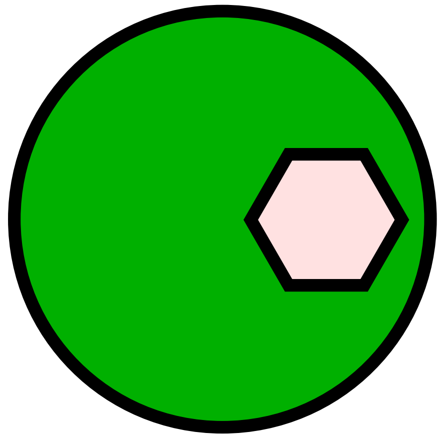
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

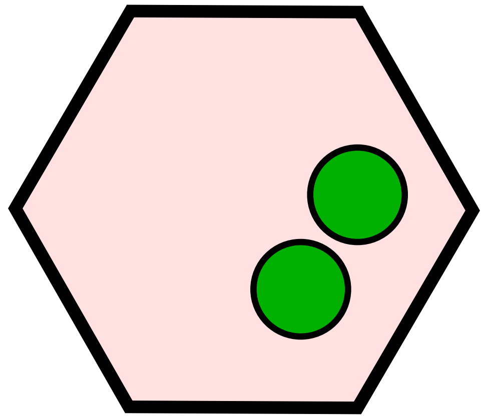
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab DISCO

  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



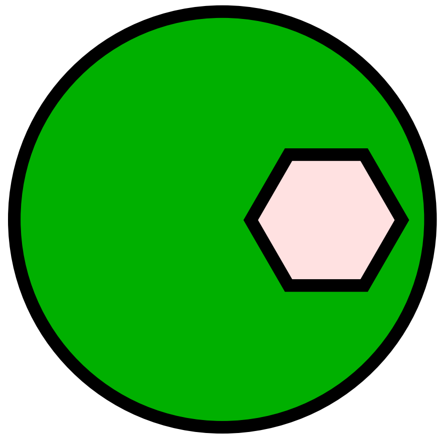
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

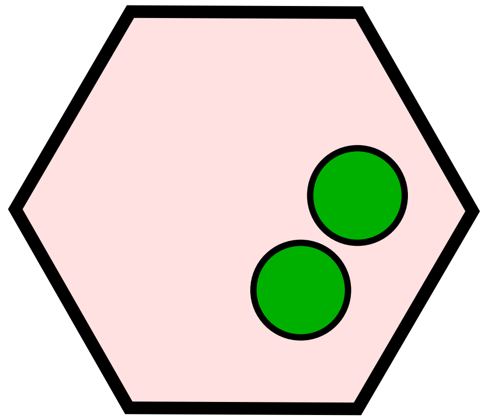
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab DISCO

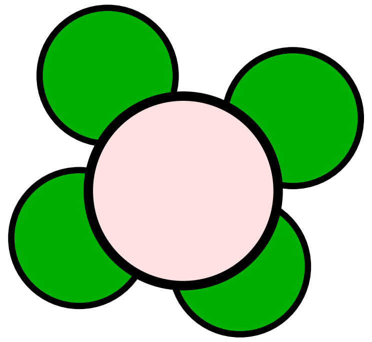
  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



- Self-contained environment

  SPM.Python

  - Maximum flexibility
  - Fewest headaches
  - Fault tolerance is inherited



Preamble: "Exploiting Parallelism"

Parallelism: The management of a collection of serial tasks

Management: The policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

Serial Tasks: Are classified in terms of either:

- Coarse grain ... where tasks may not communicate prior to conclusion, or
- Fine grain ... where tasks may communicate prior to conclusion.

Preamble: "Exploiting Parallelism"

Parallelism: The management of a collection of serial tasks

Management: The policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- ...ed, and
- the manner in which resources are obtained and released

> Management policies codify how serial tasks are to be managed ... independent of what they may be

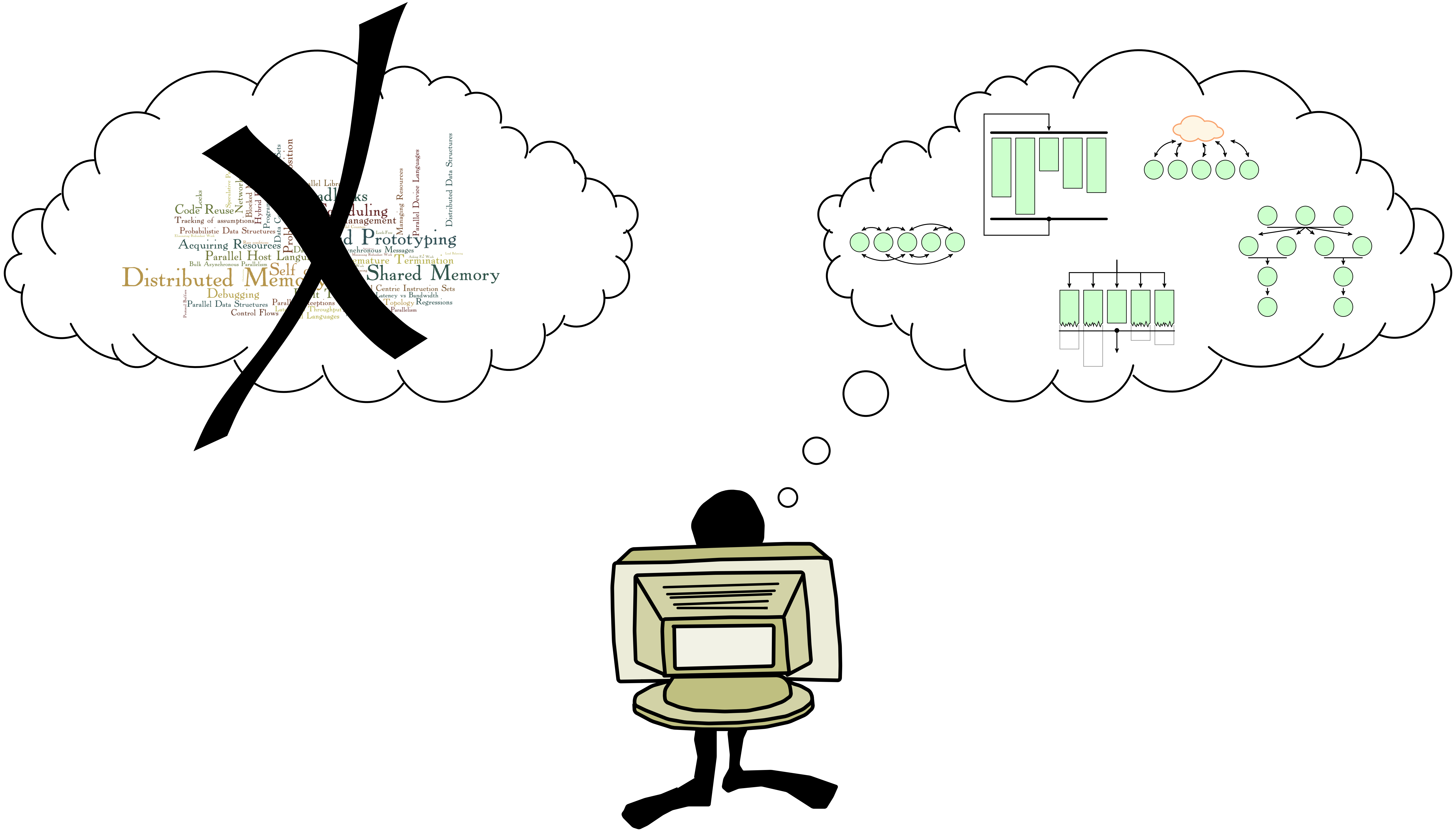
Serial Tasks: Are classified in terms of either:

- Coarse grain ... where tasks may not communicate prior to conclusion, or
- Fine grain ... where tasks may communicate prior to conclusion.

management:

... a more challenging facet of [parallel] software engineering ...

- The Future of Computing Performance: Game Over or Next Level?
National Academy of Sciences, 2011


Code Reuse
Tracking of assumptions
Probabilistic Data Structures
Acquiring Resources
Shared Memory
Debugging
Parallel Data Structures
Control Flows
Termination
Latency vs Bandwidth
Topology
Regressions
Managing Resources
Parallel Device Languages
Distributed Data Structures

Module (Serial)

Module (Parallel)

Parallel Workflow

```
def taskEval(remoteArgs):
    return util.coprocess \
               (cmd     = "%(bin)s  " \
                          "%(v)s    " \
                          "-c %(c)s " \
                          "-d %(d)s " \
                          "-s %(s)s " \
                          "-a %(a)s " \
                          "-o %(o)s " % dict(bin =          remoteArgs.bin,
                                             v   = { True  : '-v',
                                                     False : '',
                                                     None  : '',
                                                     }   [ remoteArgs.v ],
                                             c   =          remoteArgs.c,
                                             d   =          remoteArgs.d,
                                             s   =          remoteArgs.s,
                                             a   =          remoteArgs.a,
                                             o   =          remoteArgs.o,
                                             ),
                timeout = util.timeout.after(seconds = remoteArgs.timeout));
```

Module (Serial) | Module (Parallel) | Parallel Workflow

```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/2E/bin/run_I2EAWrapper.sh";
    rankD = util.prank.policyD();

    return util.coprocess \
               (cmd     = "%(bin)s       " \
                          "%(id)s        " \
                          "%(dotConfig)s " \
                          "%(dotTgz)s    " \
                          "%(rankD)s     " % dict(bin       =           bin,
                                                  id        = remoteArgs.id,
                                                  dotConfig = remoteArgs.dotConfig,
                                                  dotTgz    = remoteArgs.dotTgz,
                                                  rankD     = rankD,
                                                  ),
                timeout = util.timeout.after (seconds   = remoteArgs.timeout));
```

Module (Serial) | **Module (Parallel)** | Parallel Workflow

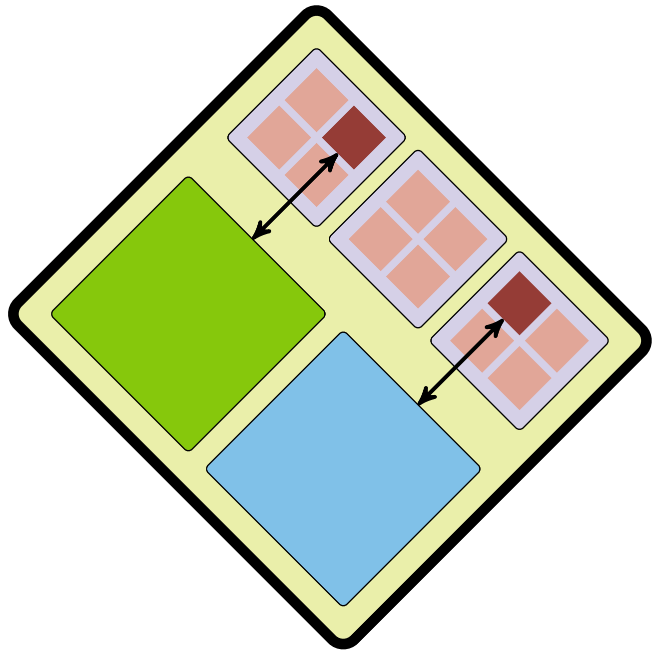
```
def main(pool, env):
     taskSubmit (env                        = env) \
    .managerEval(pool                       = pool,
                 timeoutWaitForSpokes = 2,    # Secs
                 timeoutExecution     = 300, # Secs
                );

     if (terminateEarly):
          raise Exception("phaseA failed");

@grainCoarseSingleton.pclosure
def taskSubmit():
     return stdlib.cache(# Core options
                         bin = '...',
                         v   = True,
                         c   = "/nfs/expt100000/dotCConfig.json",
                         d   = "/nfs/expt100000/dotD",
                         s   = "/nfs/expt100000/dotSConfig.json",
                         a   = "/nfs/expt100000/dotAConfig.json",
                         o   = "/nfs/expt100000/output",
                         # Meta options
                         timeout = 300, # Secs
                         label   = "phaseA (exec)",
                         env     = env);
```

```
def main(pool, env, tasks):
    tasksSubmit(env                    = env,
                tasks                  = tasks) \
   .managerEval(pool                   = pool,
                timeoutWaitForSpokes = 2,    # Secs
                timeoutExecution     = 300, # Secs
               );

    if (terminateEarly):
        raise Exception("imageToEdge failed");

@grainCoarseList.pclosure
def tasksSubmit(env, tasks):
    rval = [];
    for cmd in filter(len,                        # Skip any empty line ...
                      map((lambda x:
                              x.strip()),         # Skip any prefix/suffix spaces ...
                          tasks)):
        rval += [ stdlib.cache(cmd     = cmd,
                               timeout = 2,    # Secs
                               label   = "imageToEdge (exec - %(ct)d)" \% dict(ct = len(rval),),
                               ),
                 ];
    return rval;
```

Module (Serial) | Module (Parallel) | **Parallel Workflow**

```
def main():
    import __hidden__.pool as pool;
    import __hidden__.env  as env;
    import util.phaseA.par as phaseA;
    import util.phaseB.par as phaseB;
    import util.phaseC.par as phaseC;
    import util.phaseD.par as phaseC;

    try:
        pool = pool.interAll();
        env  = env .main    ();

        phaseA                 .main(pool = pool, env = env);
        phaseB                 .main(pool = pool, env = env);
        tasks2ndRound = phaseC.main(pool = pool, env = env);
        phaseD                 .main(pool = pool, env = env, tasks = tasks2ndRound);
    except Exception e:
        ...

    return;
```

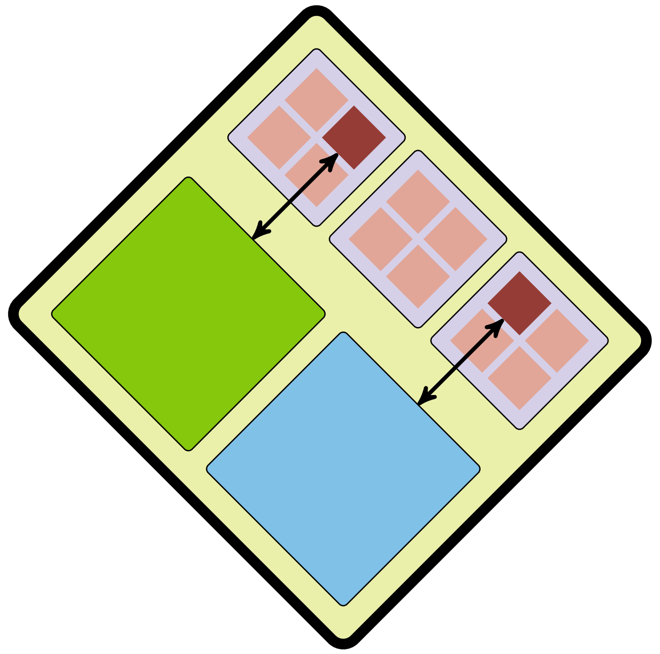
Module (Parallel): Intra-node



Device-specific Component
(in Emerald, C, C++ and Fortran)

```
def::api pow2::void <Target = Cuda>      \
                    (var a::matrixA&):
  # wrapper around implementation in C++

def::api pow2::void <Target = X98Cores> \
                    (var a::matrixA&):
  # wrapper around implementation in C
  ...

def::api pow2::void <Target = Serial>    \
                    (var a::matrixA&):
  ...
```

Module (Parallel): Intra-node



Heterogeneous Component
(in Emerald)

```
def::api main::void (dim1::int, dim2::int):
  var a::matrixA[dim1,dim2] = rand;
  var b::matrixB[dim1,dim2] = rand;

  try::concurrent:
    from  ( demo :: explict )  import pow2;

    pow2(a);
    b *= 2.0;
  except:
    raise;

  assert(a::(Cuda == Serial == X86Cores));
  assert(b::(Cuda == Serial));

  from global import result;

  result = a::Cuda;

  return;
```

ice-specific Component
rald, C, C++ and Fortran)

```
::void <Target = Cuda>          \
        (var a::matrixA&):
round implementation in C++

::void <Target = X98Cores>  \
        (var a::matrixA&):
round implementation in C

::void <Target = Serial>       \
        (var a::matrixA&):
```

Module (Parallel): Intra-node

Heterogeneous Data Structure (in Emerald)

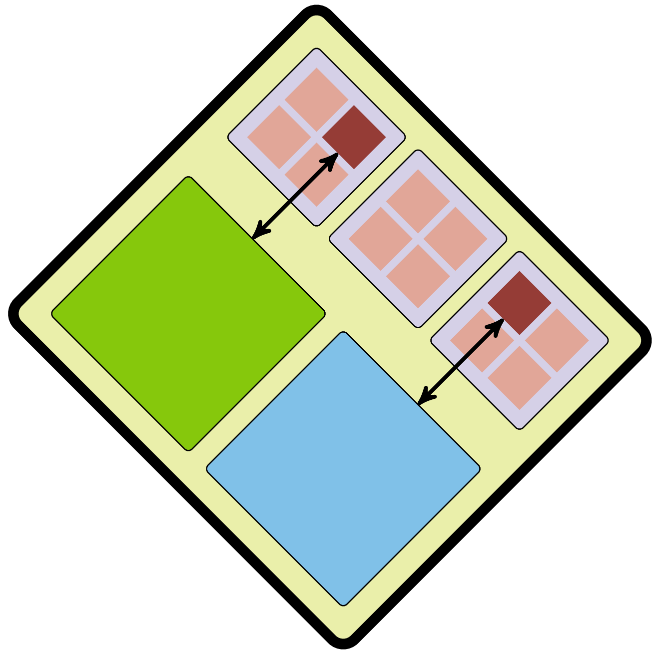
```
def::struct myMatrix (nDim1 :: int,
                      nDim2 :: int):
  - @ -::Array
    Domain = (nDim1, nDim2);
    Format = Row;
    Target = (::CUDA, ::Serial, ::X86Cores);
  - @ -
  {
    float _;
  };
```

ous Component
merald)

```
d (dim1::int, dim2::int):
im1,dim2] = rand;
im1,dim2] = rand;

    from  ( demo :: explict )  import pow2;

    pow2(a);
    b *= 2.0;
  except:
    raise;

  assert(a::(Cuda == Serial == X86Cores));
  assert(b::(Cuda == Serial));

  from global import result;

  result = a::Cuda;

  return;
```

ice-specific Component
rald, C, C++ and Fortran)

```
::void <Target = Cuda>            \
       (var a::matrixA&):
round implementation in C++

::void <Target = X98Cores> \
       (var a::matrixA&):
round implementation in C

::void <Target = Serial>    \
       (var a::matrixA&):
```



Suppositions

Most embarrassingly parallel solutions
perform a lot of redundant work ...

Suppositions

Most embarrassingly parallel solutions
perform a lot of redundant work ...

$$\Downarrow$$

- Only fix ... share knowledge

Suppositions

Many problems in HPC and Analytics are memory bounded ...

Suppositions

Many problems in HPC and Analytics are memory bounded ...

$$\Downarrow$$

- Cannot depend on virtualization
- Must throw everything at the problem

Suppositions: Constraints When Exploiting Parallelism

Prototypes
vs
Runtime Env

Suppositions: Constraints When Exploiting Parallelism

Prototypes vs Runtime Env	$\longrightarrow$	Construct-by-correction

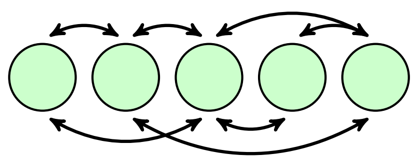
Suppositions: Constraints When Exploiting Parallelism

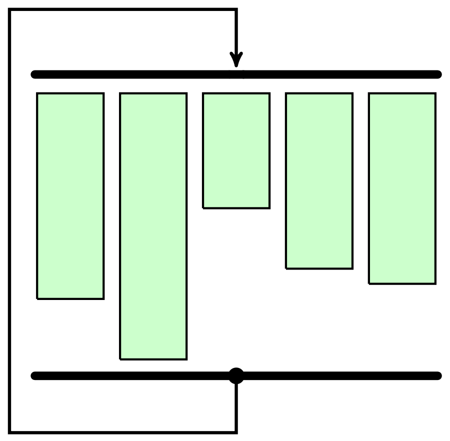
Prototypes	$\longrightarrow$	Construct-by-correction
vs		
Runtime Env	$\longrightarrow$	Correct-by-construction

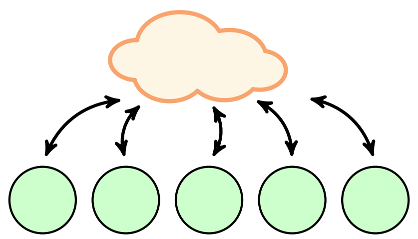
Suppositions: Enabling / Disabling Communication Primitives

SendRecvA(...)

\+





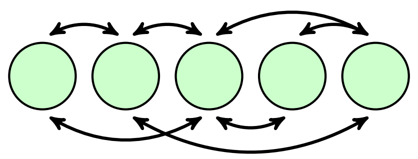


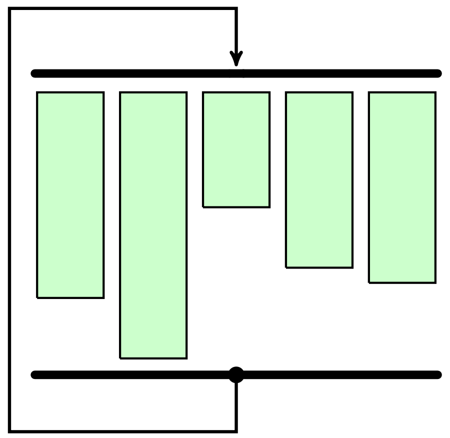
⇓	⇓	⇓
No Deadlocks	Rare Deadlocks	Frequent Deadlocks

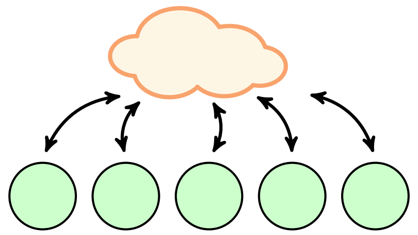
Suppositions: Parallel Semantics

division by zero

+







$\Downarrow$

No
Side-effects

$\Downarrow$

Global
Termination

$\Downarrow$

Pruning of
pending work

Conclusion: Rest of the tutorial

For each form of parallelism to be reviewed:

- What is the management policy?
- Describe a compatible communication primitive
- Describe a toxic communication primitive