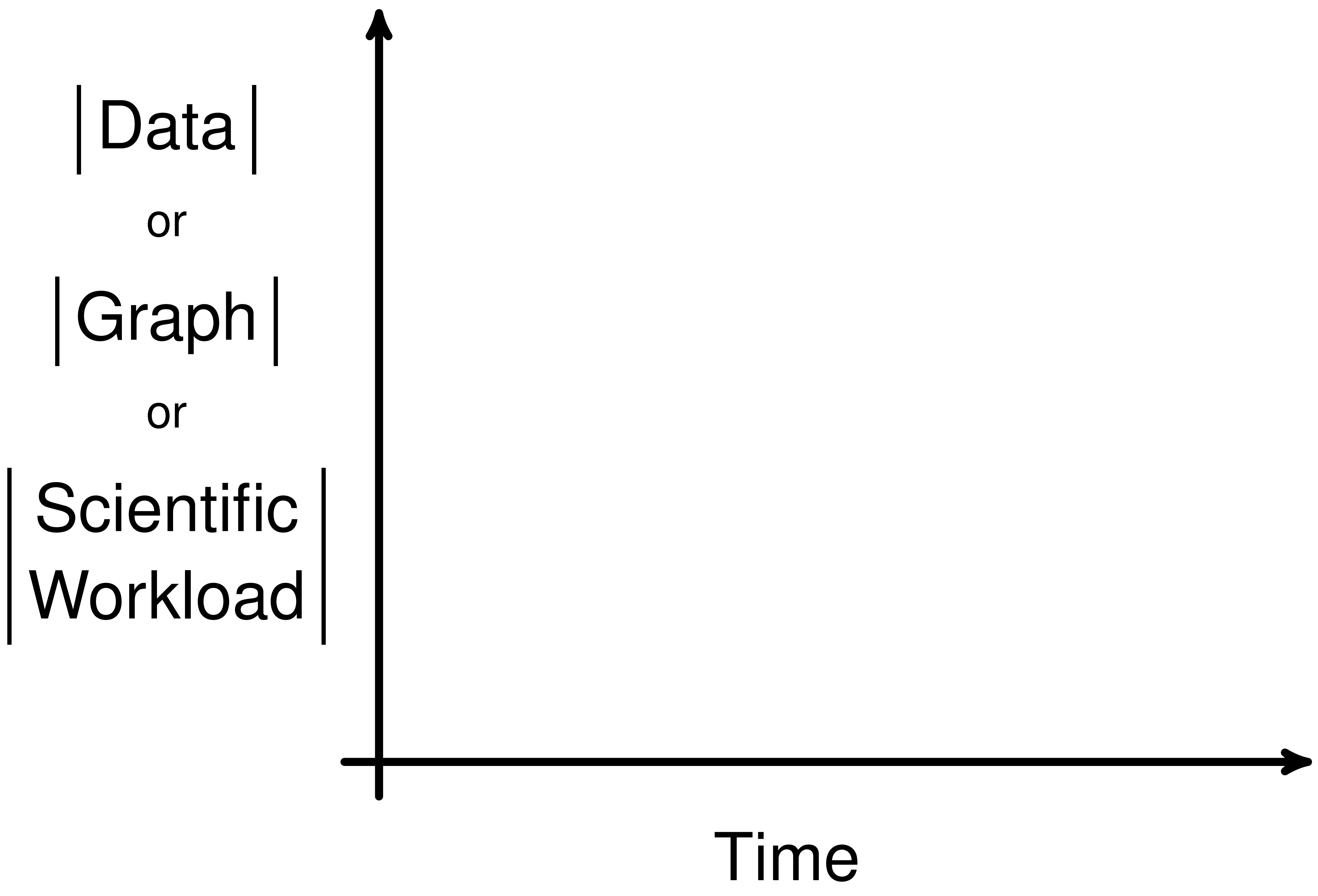
Applied Parallel Computing

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

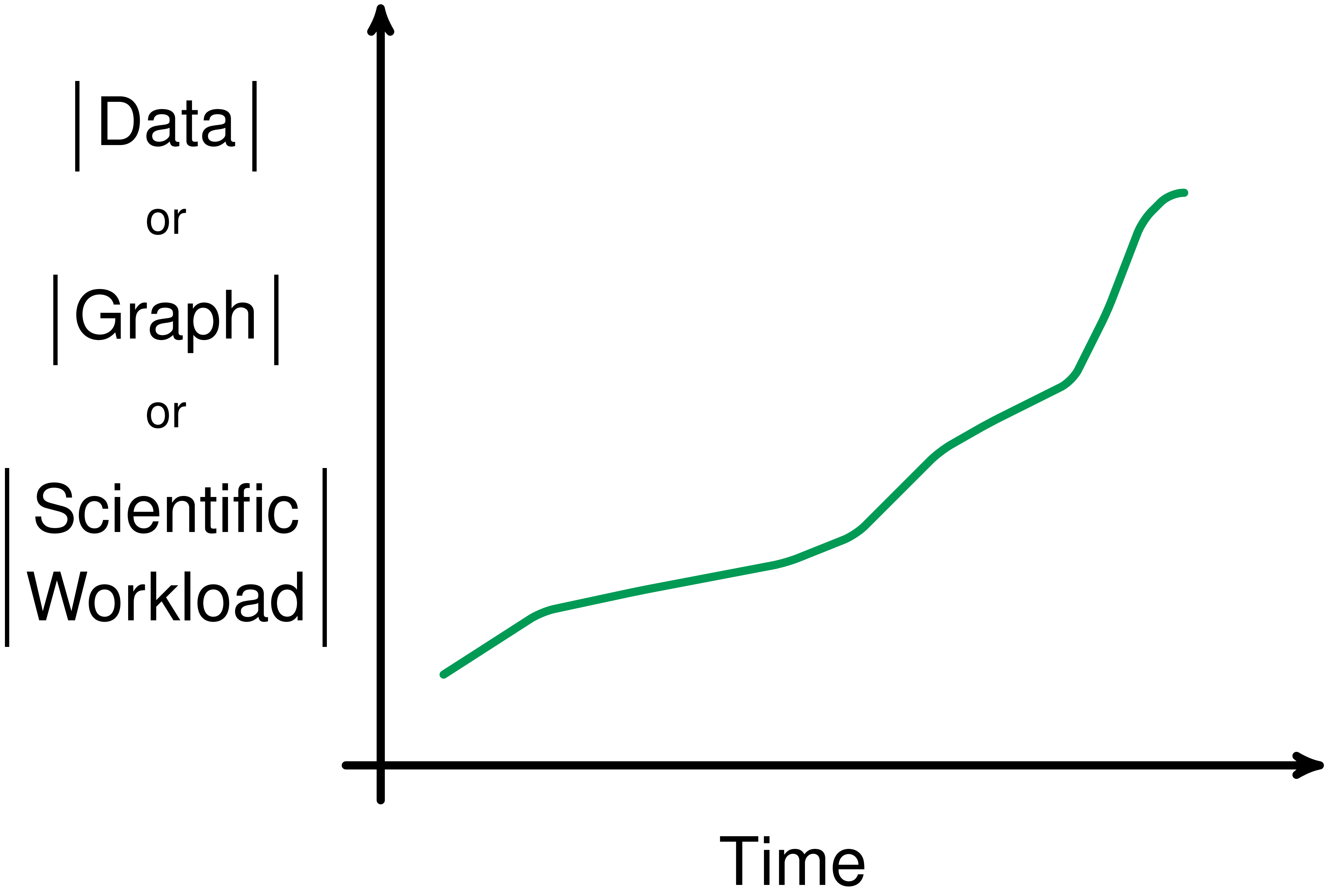
Minesh B. Amin
mamin@mbasciences.com
www.mbasciences.com

PyCON 2013
Santa Clara, CA
March 14, 2013

$\left\{\begin{array}{c}\textbf{Robust}\\\textbf{Fault tolerant}\end{array}\right\}$ Parallelism: Why?

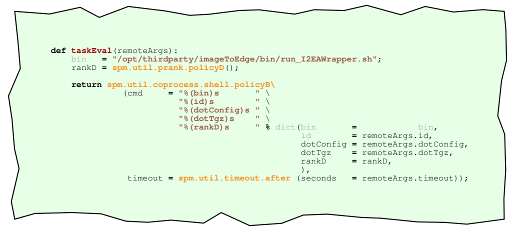


$\left\{\begin{array}{c}\textbf{Robust}\\ \textbf{Fault tolerant}\end{array}\right\}$ Parallelism: Why?



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

Serial Module

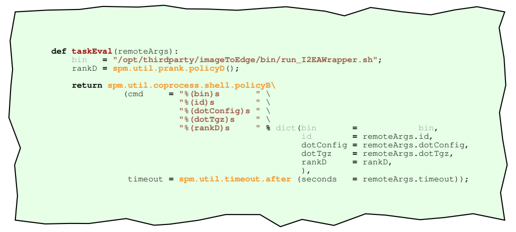


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

Robust
Fault tolerant Parallelism: How (Take I)?

Serial
Module

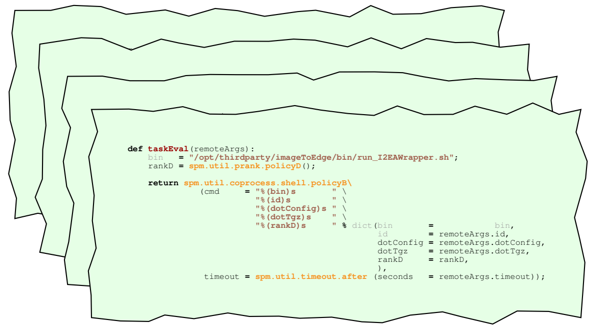


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple
Invocations

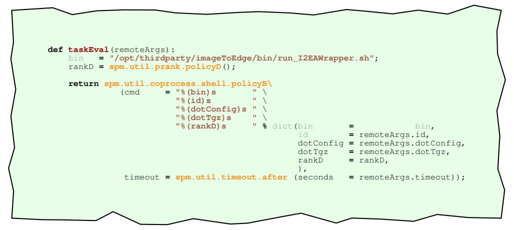


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

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

Serial
Module

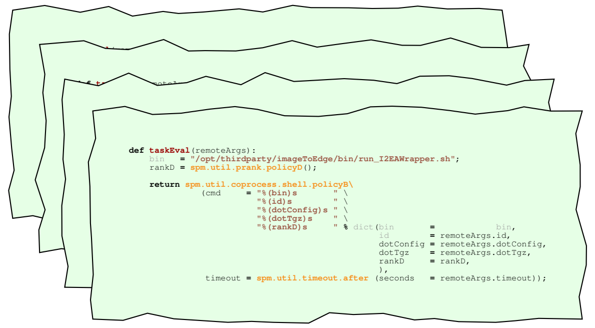


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple
Invocations

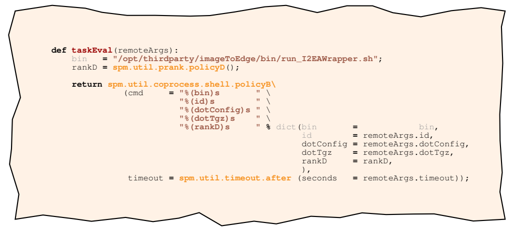


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

→

Parallel
Module

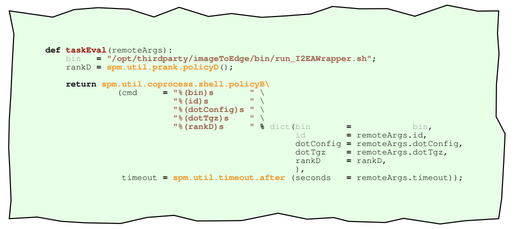


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

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

Serial Module

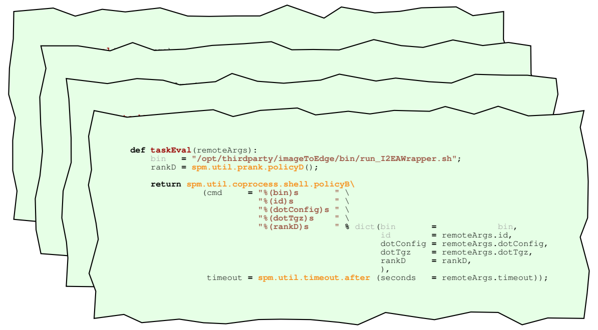


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
             (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

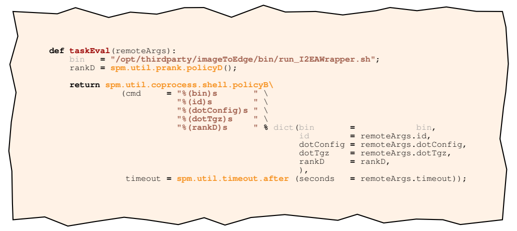
Multiple Invocations



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

    return spm.util.coprocess.shell.policyB\
             (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

→ Parallel Module

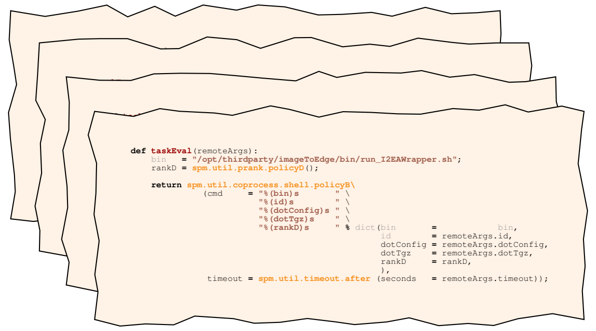


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
             (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

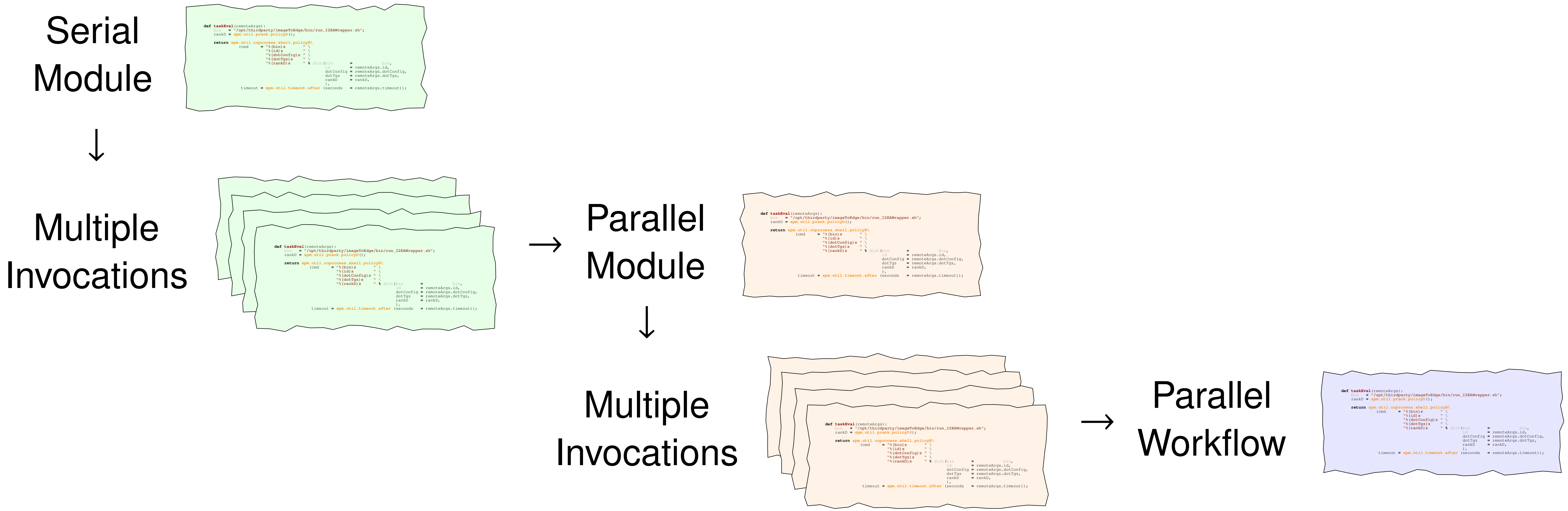
Multiple Invocations



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

    return spm.util.coprocess.shell.policyB\
             (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

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

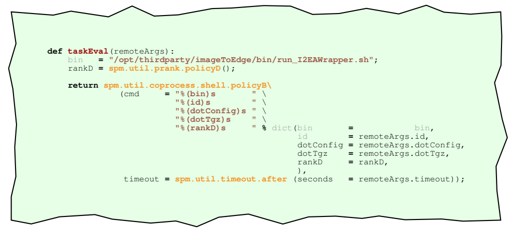


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
               (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

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

Serial Module

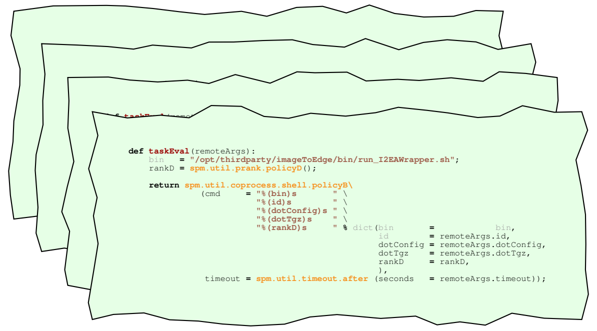


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

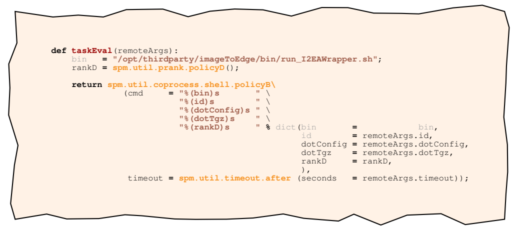
Multiple Invocations



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

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

→ Parallel Module

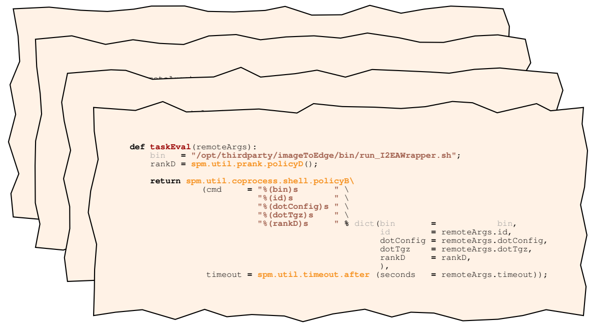


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

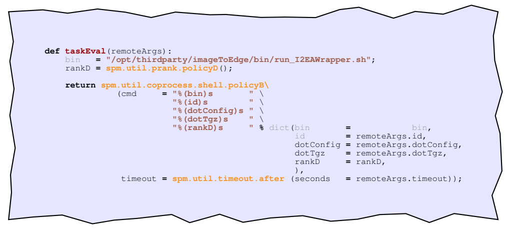
Multiple Invocations



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

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

→ Parallel Workflow

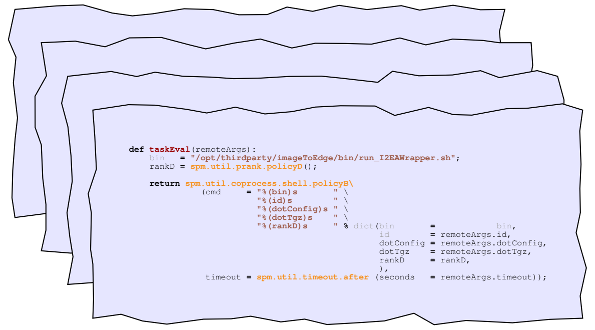


```
def taskEval(remoteArgs):
    bin   = "/opt/thirdparty/imageToEdge/bin/run_I2EAWrapper.sh";
    rankD = spm.util.prank.policyD();

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

↓

Multiple Invocations



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

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

$\left\{\begin{array}{c}\textbf{Robust}\\ \textbf{Fault tolerant}\end{array}\right\}$ 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)?

A $\left\{\begin{array}{c}\text{Language}\\ \\ \\ \end{array}\right\}$ 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)?

A $\left\{\begin{array}{c}\text{Language}\\ \text{Runtime Env}\\ \text{Framework}\\ \\ \end{array}\right\}$ 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 $\left\{ \begin{array}{c} \text{\sout{easy}} \\ \text{hard} \end{array} \right\}$?

Copyright 1994, The UNIX-HATERS Handbook

Preamble: "The Big Picture"

What makes

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

Copyright 1994, The UNIX-HATERS Handbook

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

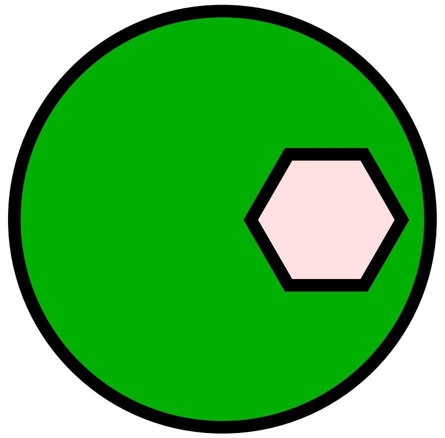
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

OpenMPI OpenMP
CUDA OpenGL

- Maximum flexibility
- Maximum headaches
- Must implement fault tolerance



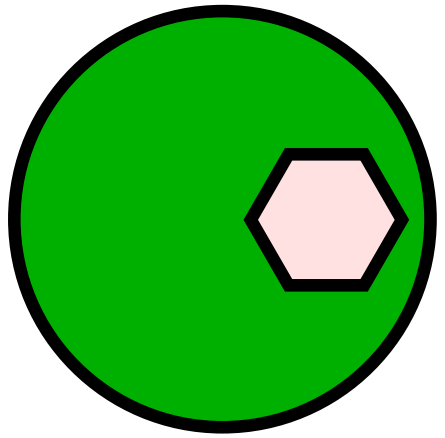
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

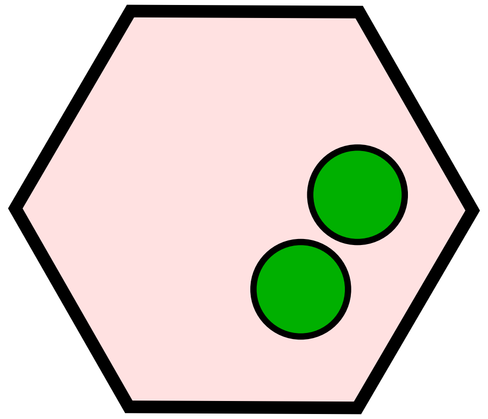
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab

  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



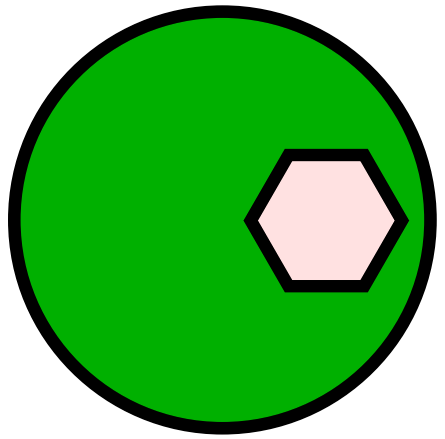
Preamble: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

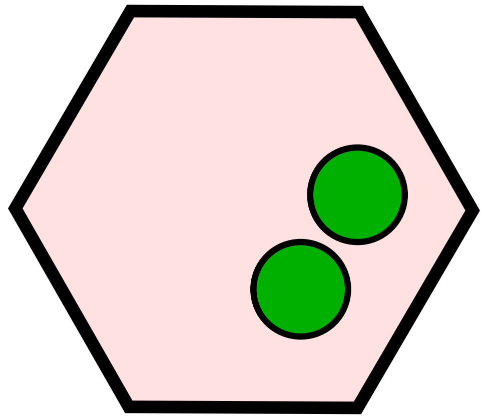
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab

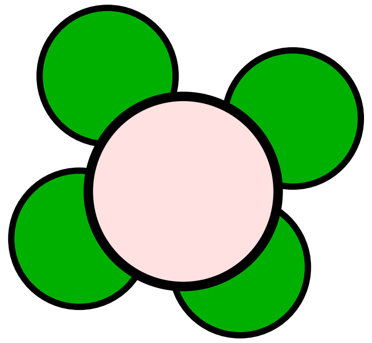
  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



- 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

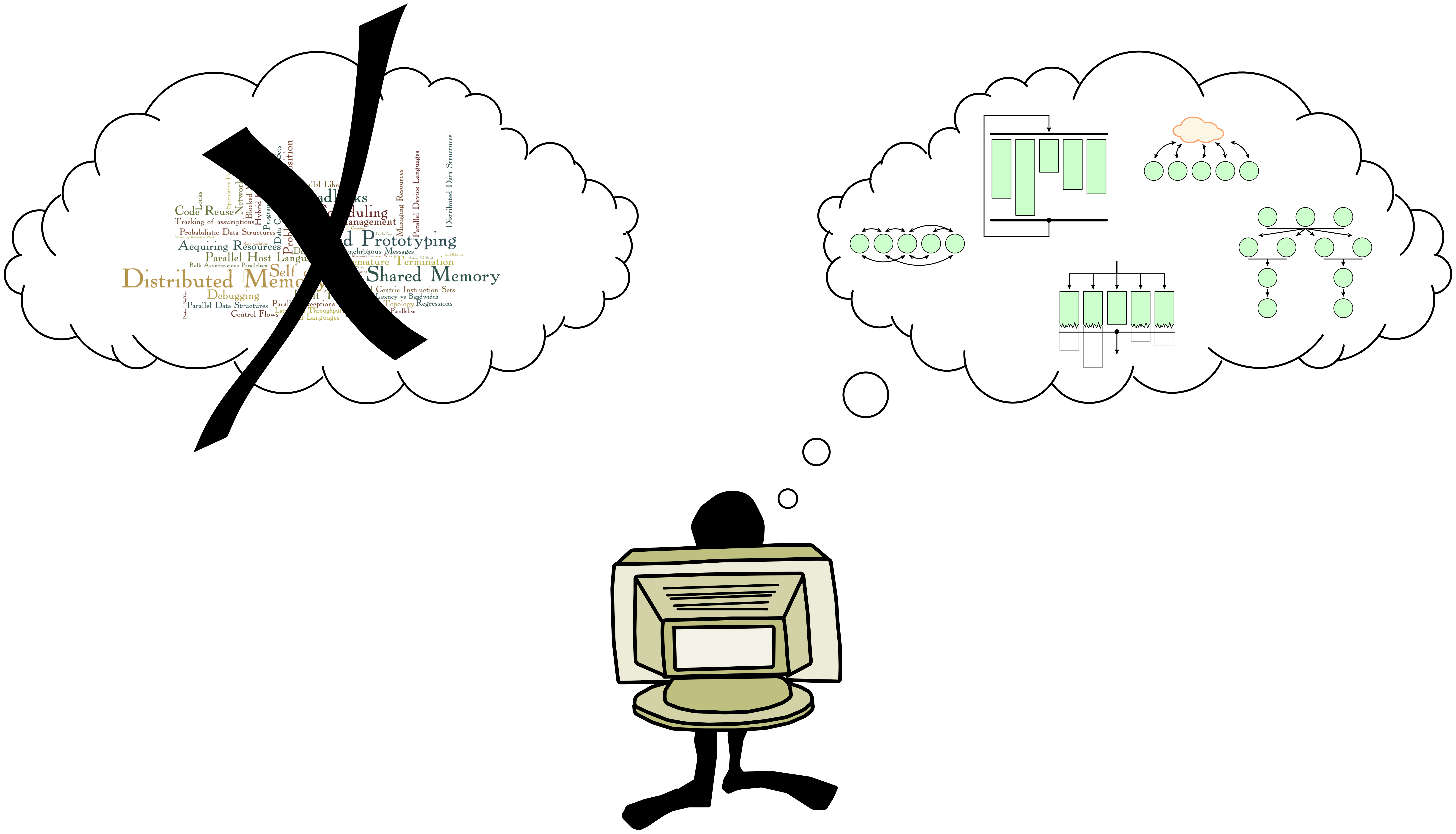
Serial Tasks: Are classified in terms of either:

- 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


Distributed Data Structures
Parallel Device Languages
Managing Resources
Locks
Speculative
Code Reuse
Blocked
Hybrid
Tracking of assumptions
Probabilistic Data Structures
Eliminating Redundant Work
Race conditions
Acquiring Resources
Lock-Free
Prototyping
Messages
Minimizing Redundant Work
Asking For Work
Load Balancing
Parallel Host
Termination
Bulk Asynchronous Parallelism
Self
Distributed
Shared Memory
Centric Instruction Sets
Latency vs Bandwidth
Debugging
Protocol Buffers
Parallel Data Structures
Topology
Regressions
Throughput
Parallelism
Control Flows
Languages

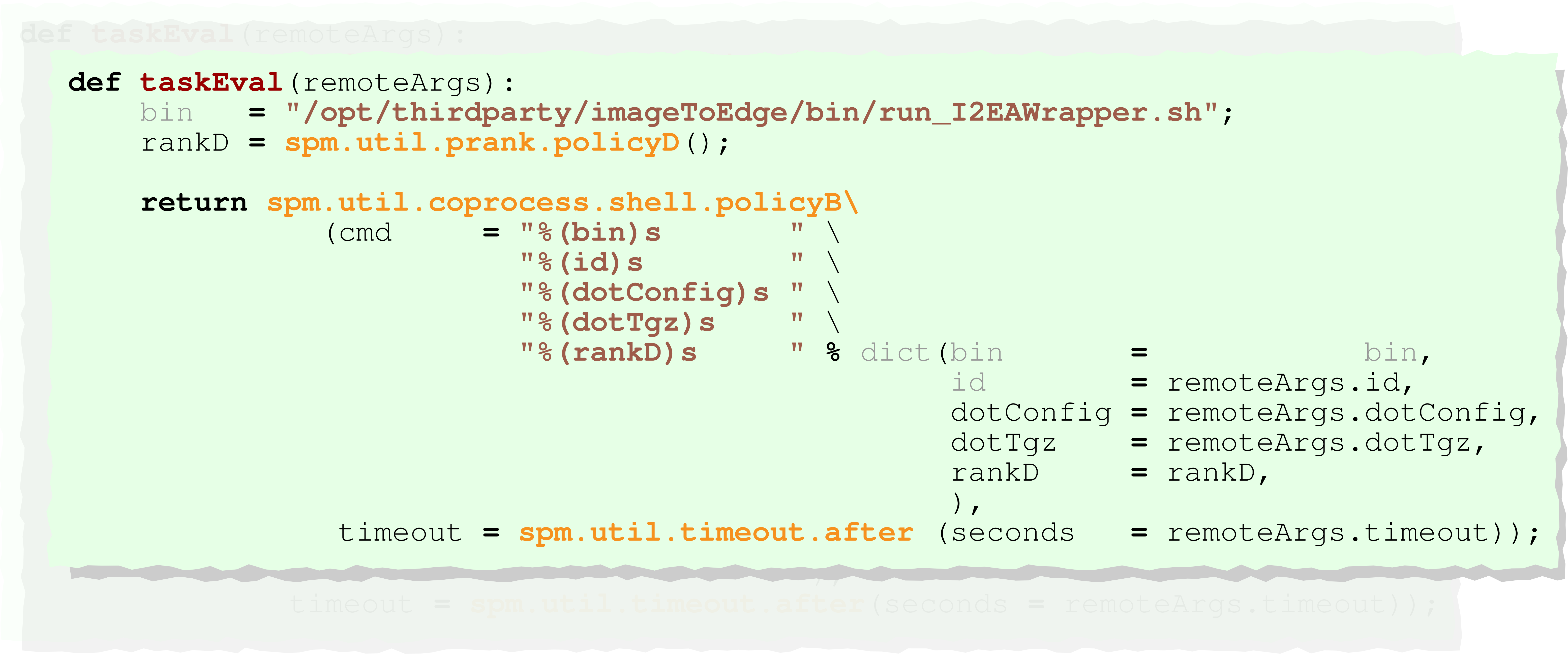
Module (Serial) Module (Parallel) Parallel Workflow

```
def taskEval(remoteArgs):
    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after(seconds = remoteArgs.timeout));
```

Module (Serial)

Module (Parallel)

Parallel Workflow



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

    return spm.util.coprocess.shell.policyB\
              (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 = spm.util.timeout.after (seconds   = remoteArgs.timeout));
```

```
def taskEval(remoteArgs):
                timeout = spm.util.timeout.after(seconds = remoteArgs.timeout));
```

Module (Serial)

Module (Parallel)

Parallel Workflow

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

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

@grainCoarseSingleton.pclosure
def submitTask():
    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);
```

Module (Serial)

Module (Parallel)

Parallel Workflow

```
def main(pool, env, batchFile):
    submitTask(env                 = env,
               batchFile            = batchFile) \
   .execTask  (pool                 = pool,
               timeoutWaitForSpokes = 2,    # Secs
               timeoutExecution     = 300, # Secs
              );

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

@grainCoarseList.pclosure
def submitTask(env, batchFile):
    rval = [];
    for cmd in filter(len,                 # Skip any empty line ...
                      map((lambda x:
                              x.strip()), # Skip any prefix/suffix spaces ...
                          batchFile.split('\n'))):
        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);
        batchFile = phaseC.main(pool = pool, env = env);
        phaseD            .main(pool = pool, env = env, batchFile = batchFile);
    except Exception e:
        ...

    return;
```

Module (Serial)

Module (Parallel)

Parallel Workflow

```
def main():
    import __hidden__.pool as pool;
    import __hidden__.env  as env;
    import util.mc    .par  as mc;

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

        mc.main(pool = pool, env  = env);
    except Exception e:
        ...

    return;
```

```
def main():

                                              env   = env);
                                              env = env);
                                              env = env);
                                              env = env, batchFile = batchFile);
    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 → Construct-by-correction

vs

Runtime Env

Suppositions: Constraints When Exploiting Parallelism

Prototypes → Construct-by-correction

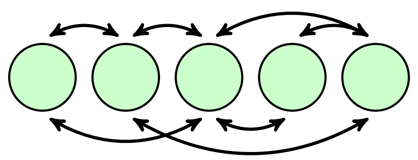
vs

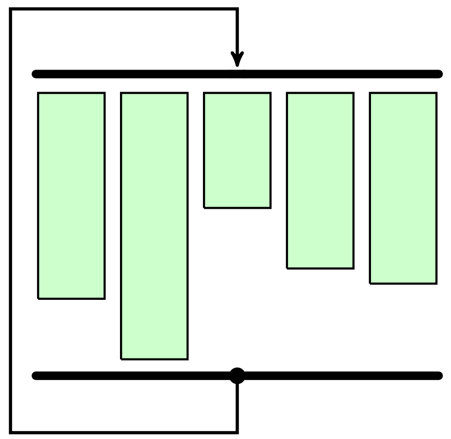
Runtime Env → Correct-by-construction

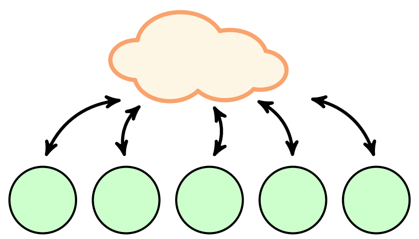
Suppositions: Enabling / Disabling Communication Primitives

SendRecvA(...)

+



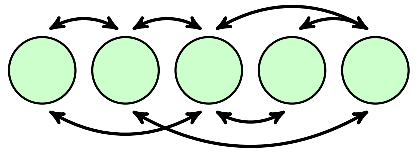




Suppositions: Enabling / Disabling Communication Primitives

SendRecvA(...)

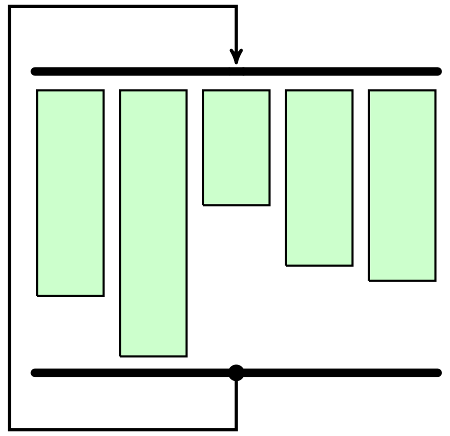
+

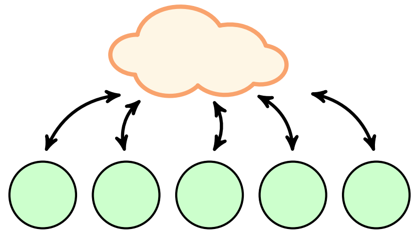


⇓

No

Deadlocks

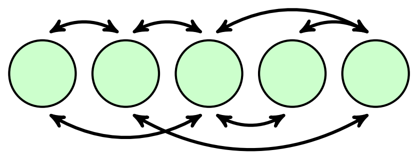


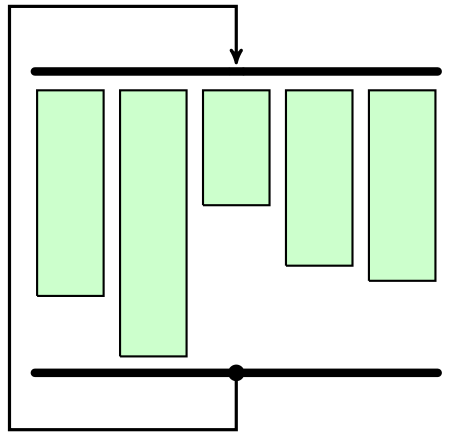


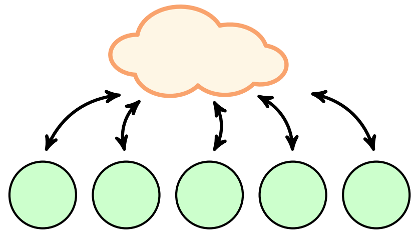
Suppositions: Enabling / Disabling Communication Primitives

SendRecvA(...)

+







⇓

No

Deadlocks

⇓

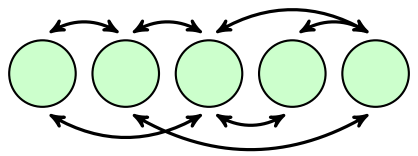
Rare

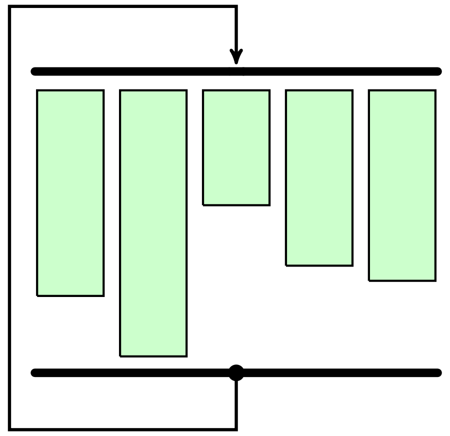
Deadlocks

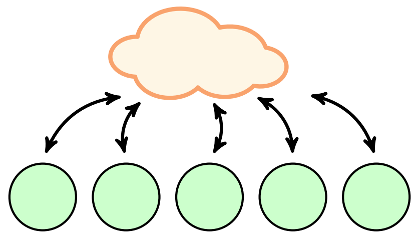
Suppositions: Enabling / Disabling Communication Primitives

SendRecvA(...)

+







⇓
No
Deadlocks

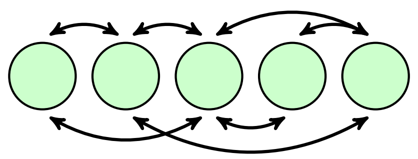
⇓
Rare
Deadlocks

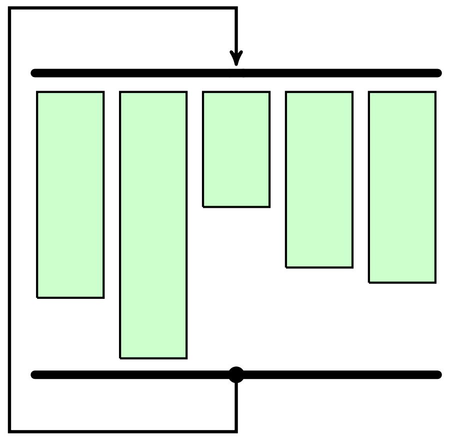
⇓
Frequent
Deadlocks

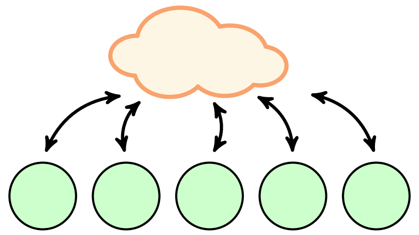
Suppositions: Parallel Semantics

division by zero

+



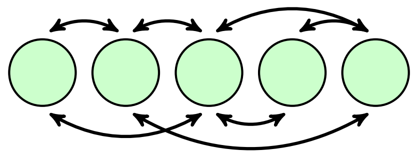


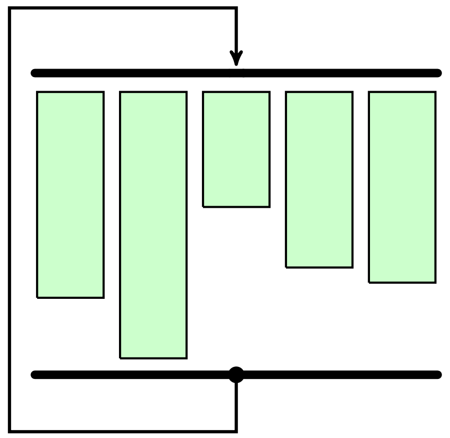


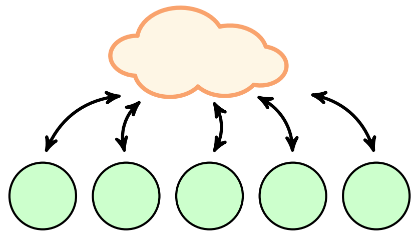
Suppositions: Parallel Semantics

division by zero

+







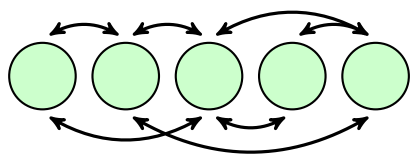
⇓

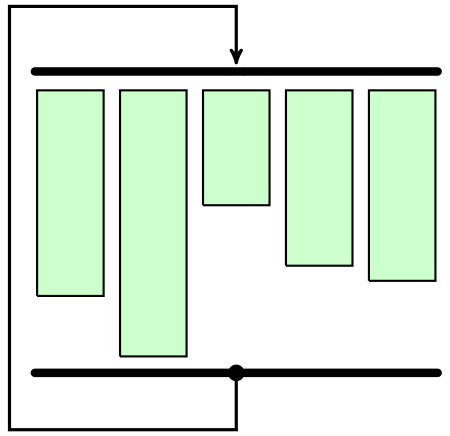
No
Side-effects

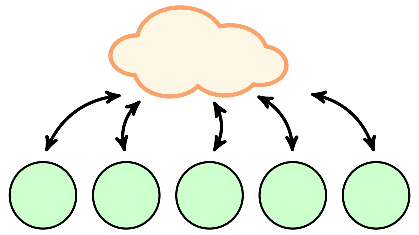
Suppositions: Parallel Semantics

division by zero

+







⇓

No Side-effects

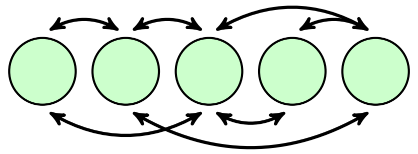
⇓

Global Termination

Suppositions: Parallel Semantics

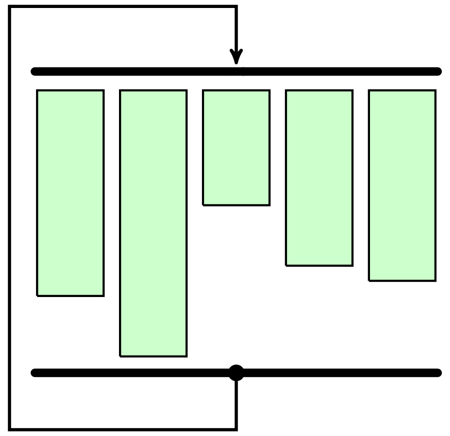
division by zero

+



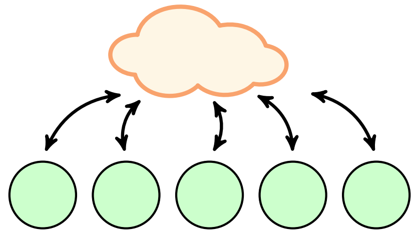
⇓

No
Side-effects



⇓

Global
Termination



⇓

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