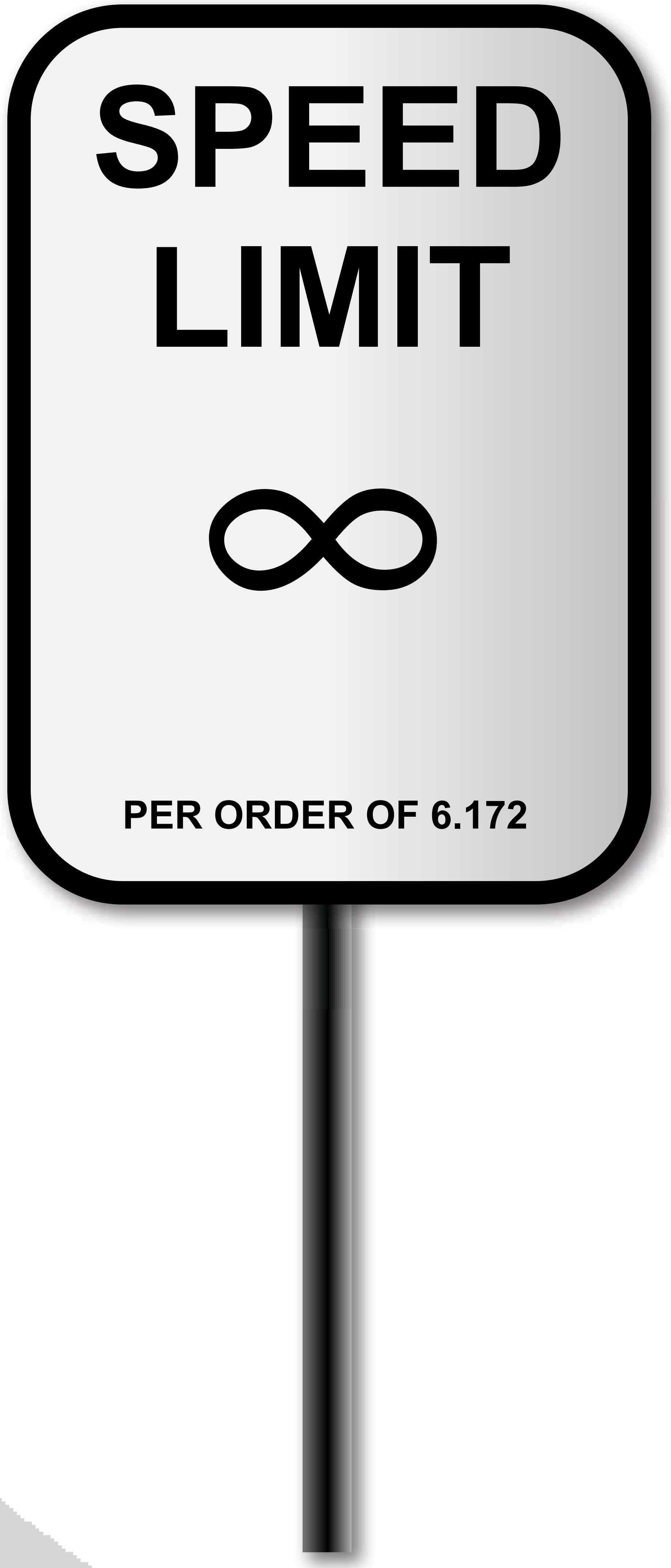
6.172
Performance
Engineering of
Software
Systems



LECTURE 18
Domain Specific Languages and Autotuning

Saman Amarasinghe

Domain Specific Languages

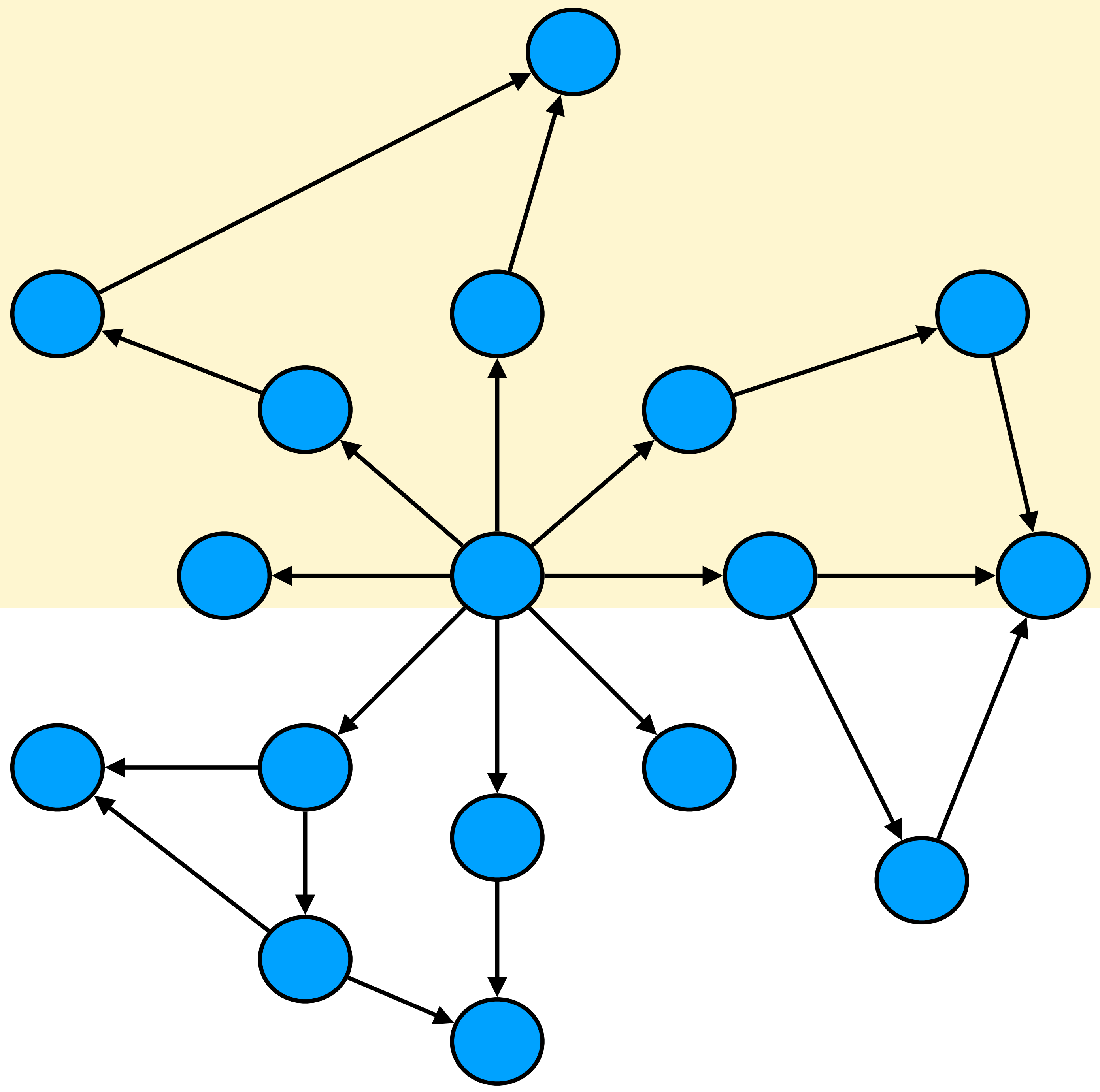
- Capture the programer intent at a higher level of abstraction
- Obtain many software engineering benefits
  - clarity, portability, maintainability, testability, etc…
- Provide the compiler more opportunities for higher performance
  - Can encode expert knowledge of domain specific transformations
  - Better view of the computation performed without heroic analysis
  - Less low-level decisions by the programmer that has to be undone

Outline

- GraphIt
- Halide
- OpenTuner

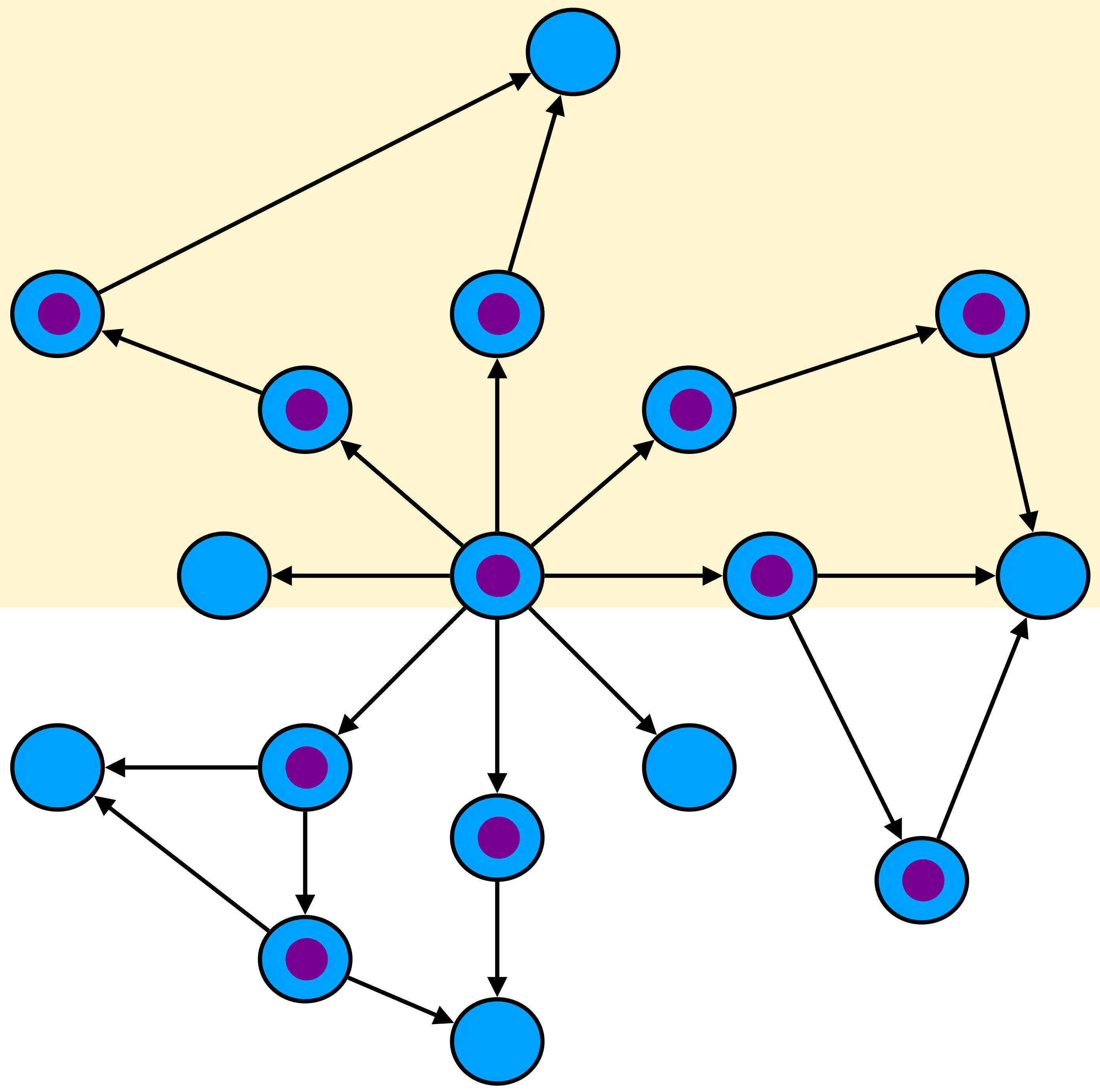
PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
    for (i = 0; i < max_iter; i++) {
        for (src : graph.vertices()) {
            for (dst : graph.getOutgoingNeighbors(node)) {
                new_rank[dst] += old_rank[src]/out_degree[src]; } }
    for (node : graph.vertices()) {
        new_rank[node] = base_score + damping*new_rank[node]; }
    swap (old_rank, new_rank); }
}
```



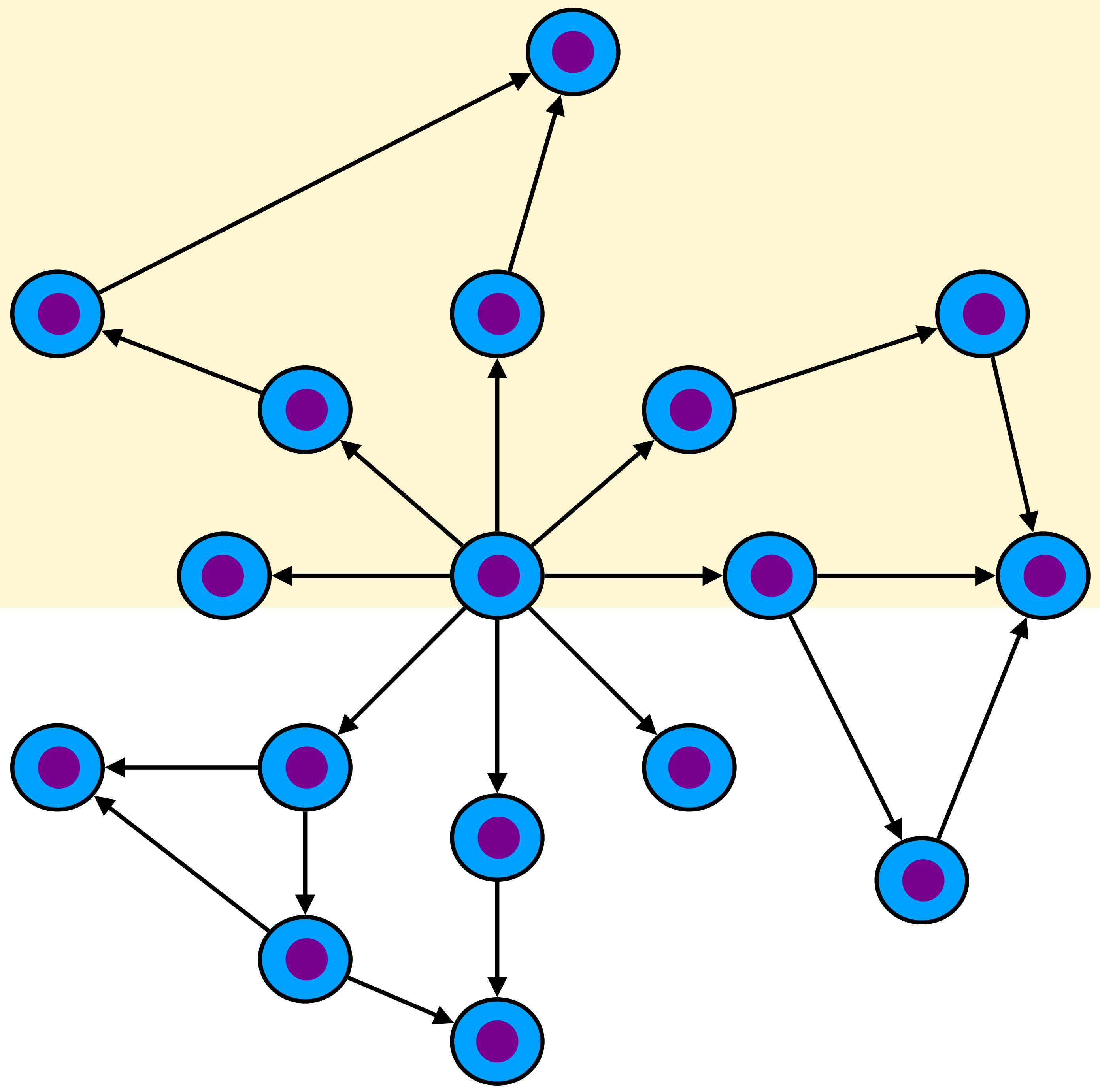
PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
  for (i = 0; i < max_iter; i++) {
    for (src : graph.vertices()) {
      for (dst : graph.getOutgoingNeighbors(node)) {
        new_rank[dst] += old_rank[src]/out_degree[src]; } }
   for (node : graph.vertices()) {
     new_rank[node] = base_score + damping*new_rank[node]; }
   swap (old_rank, new_rank); }
}
```



PageRank Example in C++

```
void pagerank(Graph &graph, double * new_rank, double * old_rank, int * out_degree, int max_iter){
   for (i = 0; i < max_iter; i++) {
      for (src : graph.vertices()) {
         for (dst : graph.getOutgoingNeighbors(node)) {
            new_rank[dst] += old_rank[src]/out_degree[src]; } }
   for (node : graph.vertices()) {
      new_rank[node] = base_score + damping*new_rank[node]; }
   swap (old_rank, new_rank); }
}
```



Hand-Optimized C++

```
template<typename APPLY_FUNC>
void edgeset_apply_pull_parallel(Graph &g, APPLY_FUNC apply_func) {
    int64_t numVertices = g.num_nodes(), numEdges = g.num_edges();
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            local_new_rank[socketId][n] = new_rank[n]; } }
    int numPlaces = omp_get_num_places();
    int numSegments = g.getNumSegments("s1");
    int segmentsPerSocket = (numSegments + numPlaces - 1) / numPlaces;
        #pragma omp parallel num_threads(numPlaces) proc_bind(spread){
        int socketId = omp_get_place_num();
        for (int i = 0; i < segmentsPerSocket; i++) {
            int segmentId = socketId + i * numPlaces;
            if (segmentId >= numSegments) break;
            auto sg = g.getSegmentedGraph(std::string("s1"), segmentId);
            #pragma omp parallel num_threads(omp_get_place_num_procs(socketId)) proc_bind(close){
                #pragma omp for schedule(dynamic, 1024)
                for (NodeID localId = 0; localId < sg->numVertices; localId++) {
                    NodeID d = sg->graphId[localId];
                    for (int64_t ngh = sg->vertexArray[localId]; ngh < sg->vertexArray[localId + 1]; ngh++) {
                        NodeID s = sg->edgeArray[ngh];
                        local_new_rank[socketId][d] += contrib[s]; }}}}}
    parallel_for(int n = 0; n < numVertices; n++) {
        for (int socketId = 0; socketId < omp_get_num_places(); socketId++) {
            new_rank[n] += local_new_rank[socketId][n]; }}}
struct updateVertex {
    void operator() (NodeID v) {
        double old_score = old_rank[v];
        new_rank[v] = (beta_score + (damp * new_rank[v]));
        error[v] = fabs((new_rank[v] - old_rank[v])) ;
        old_rank[v] = new_rank[v];
        new_rank[v] = ((float) 0) ; }; };
void pagerank(Graph &g, double *new_rank, double *old_rank, int *out_degree, int max_iter) {
    for (int i = (0); i < (max_iter); i++) {
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            contrib[v] = (old_rank[v] / out_degree[v]);};
        edgeset_apply_pull_parallel(edges, updateEdge());
        parallel_for(int v_iter = 0; v_iter < builtin_getVertices(edges); v_iter ++) {
            updateVertex()(v_iter); }; }
```

More than 23x faster

Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores.

Multi-Threaded

Load Balanced

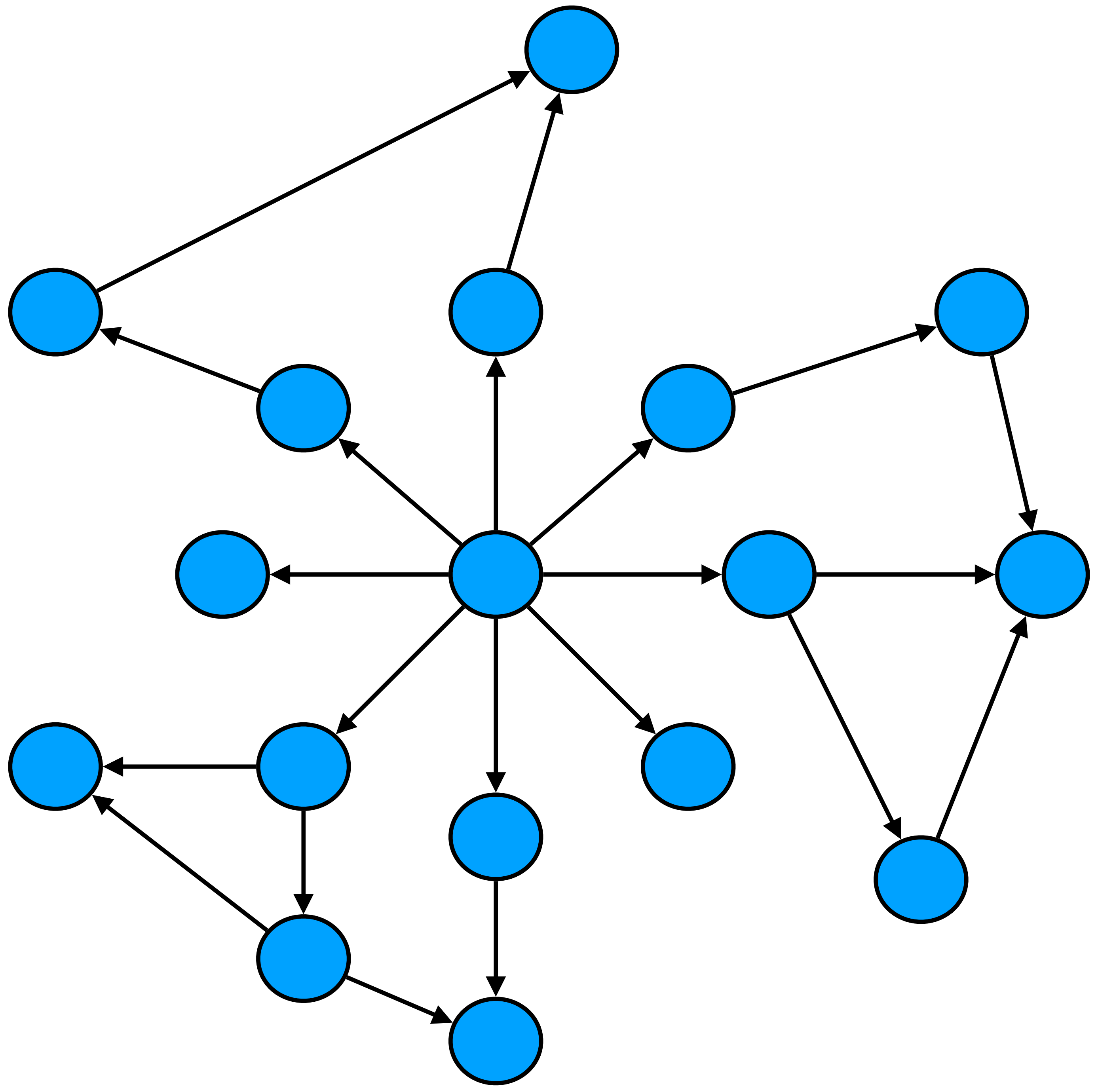
NUMA Optimized

Cache Optimized

(1) Hard to write correctly.

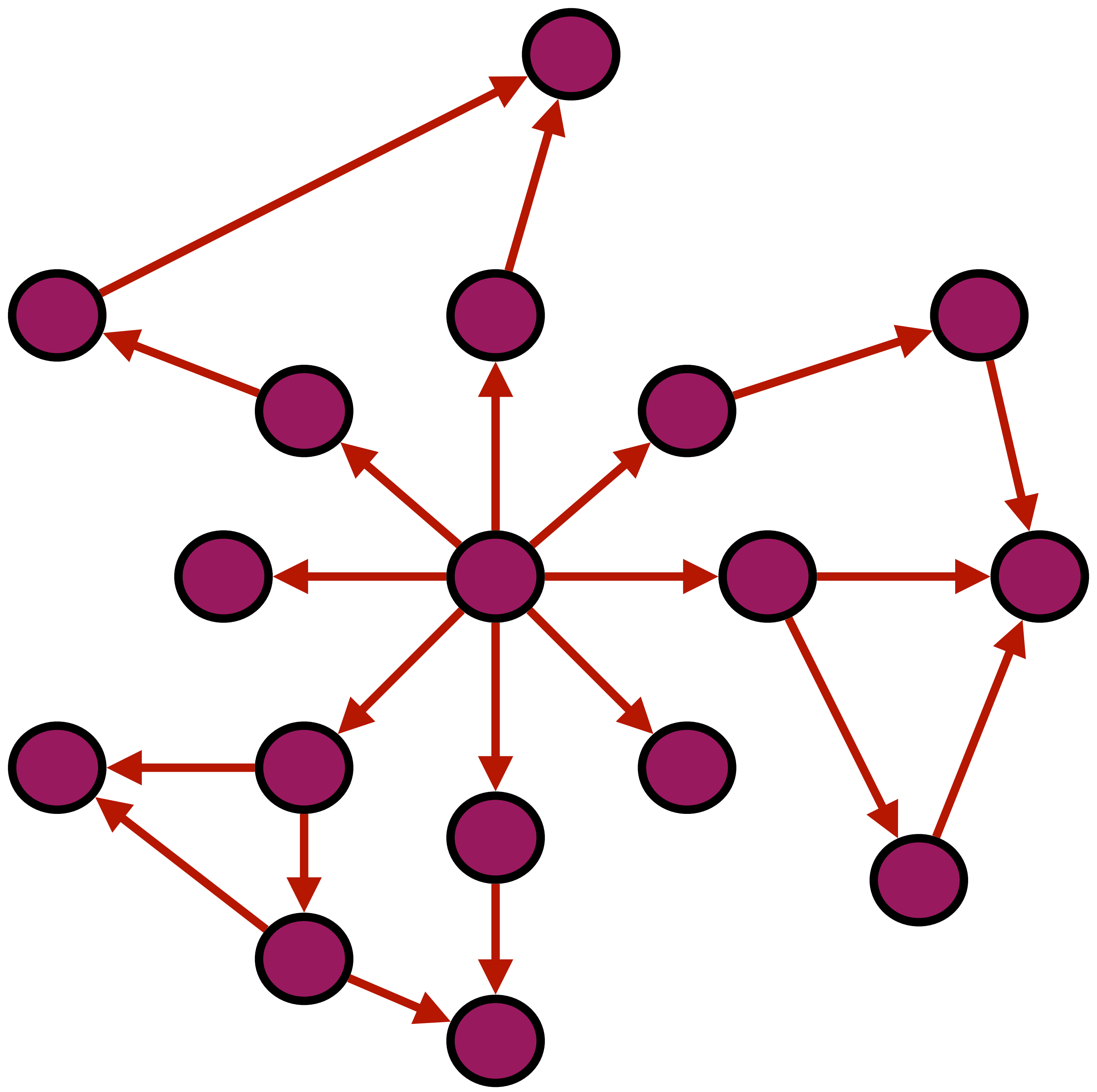
(2) Extremely difficult to experiment with different combinations of optimizations

Graph Algorithms



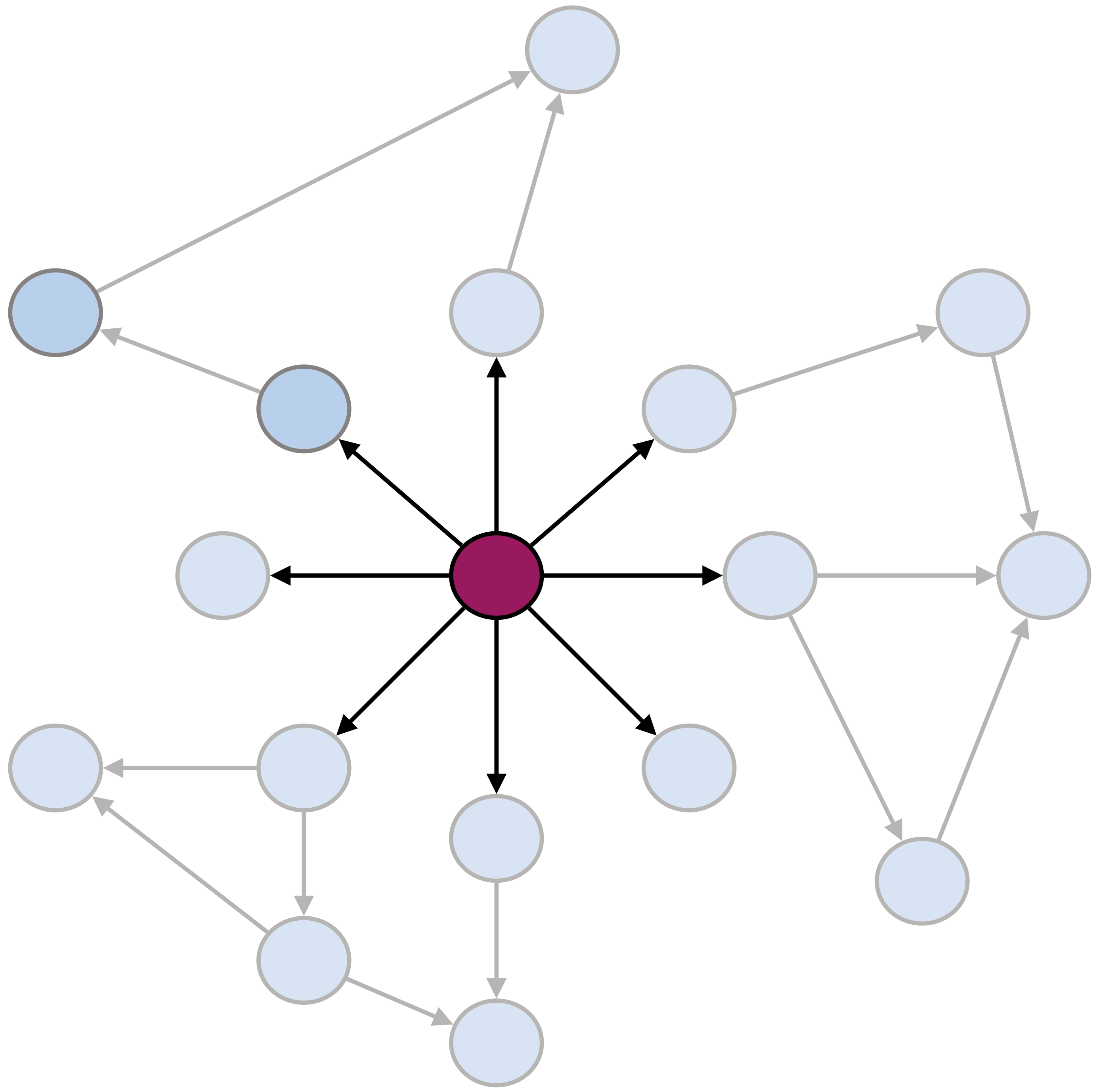
- What are the different types of graph algorithms

Topology-Driven Algorithms

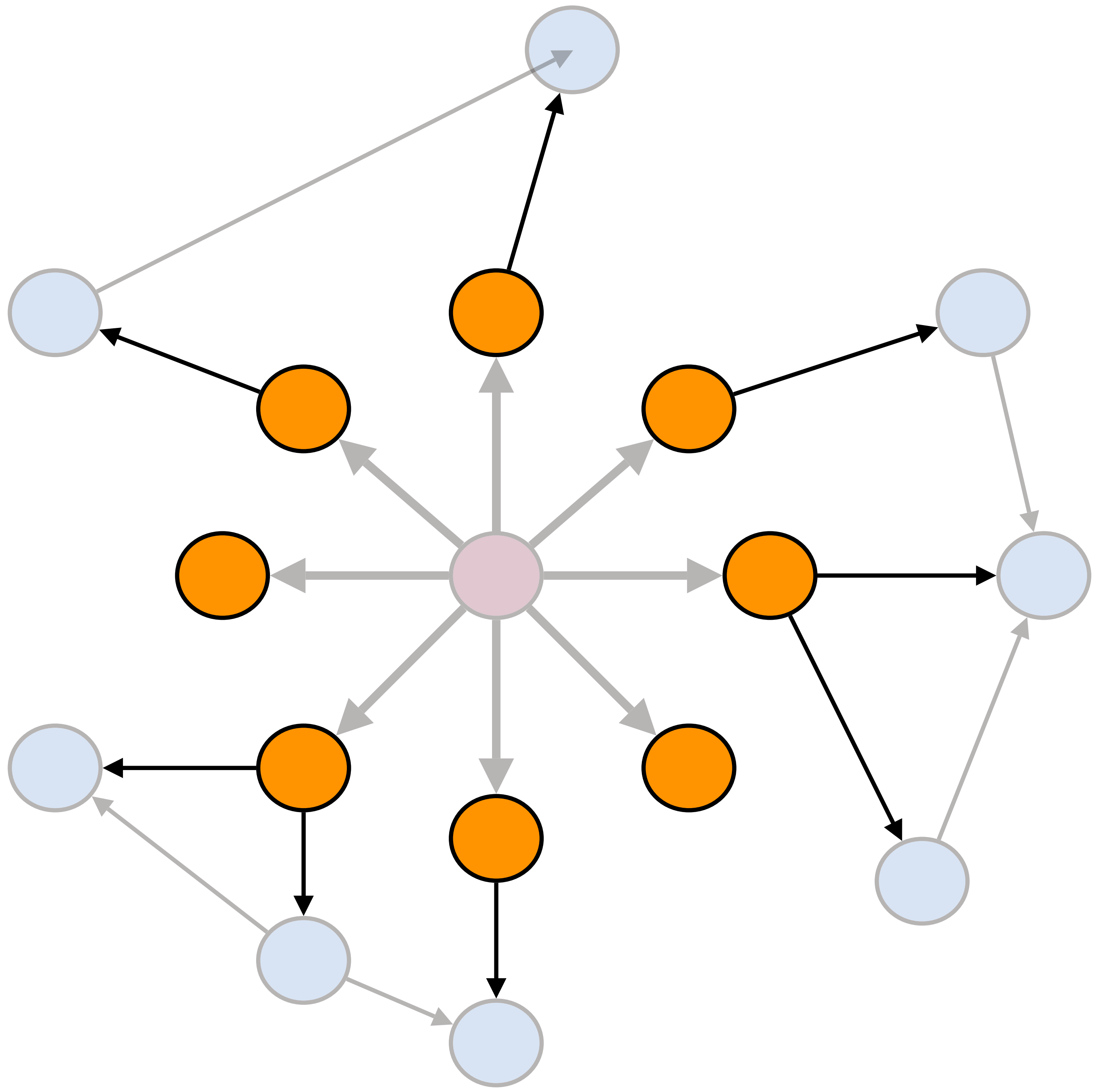


Work on All Edges and Vertices

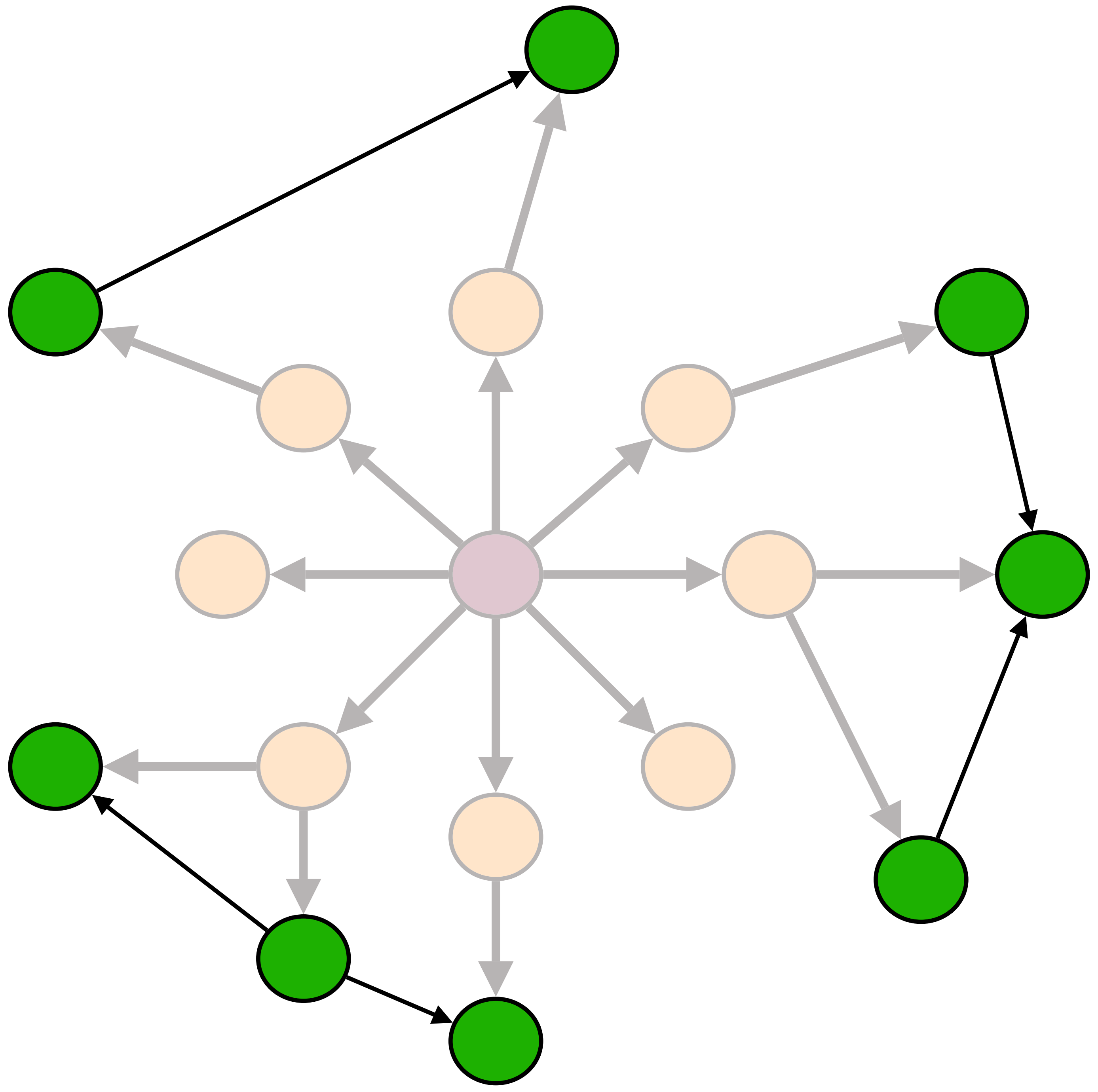
Data-Driven Algorithms



Data-Driven Algorithms

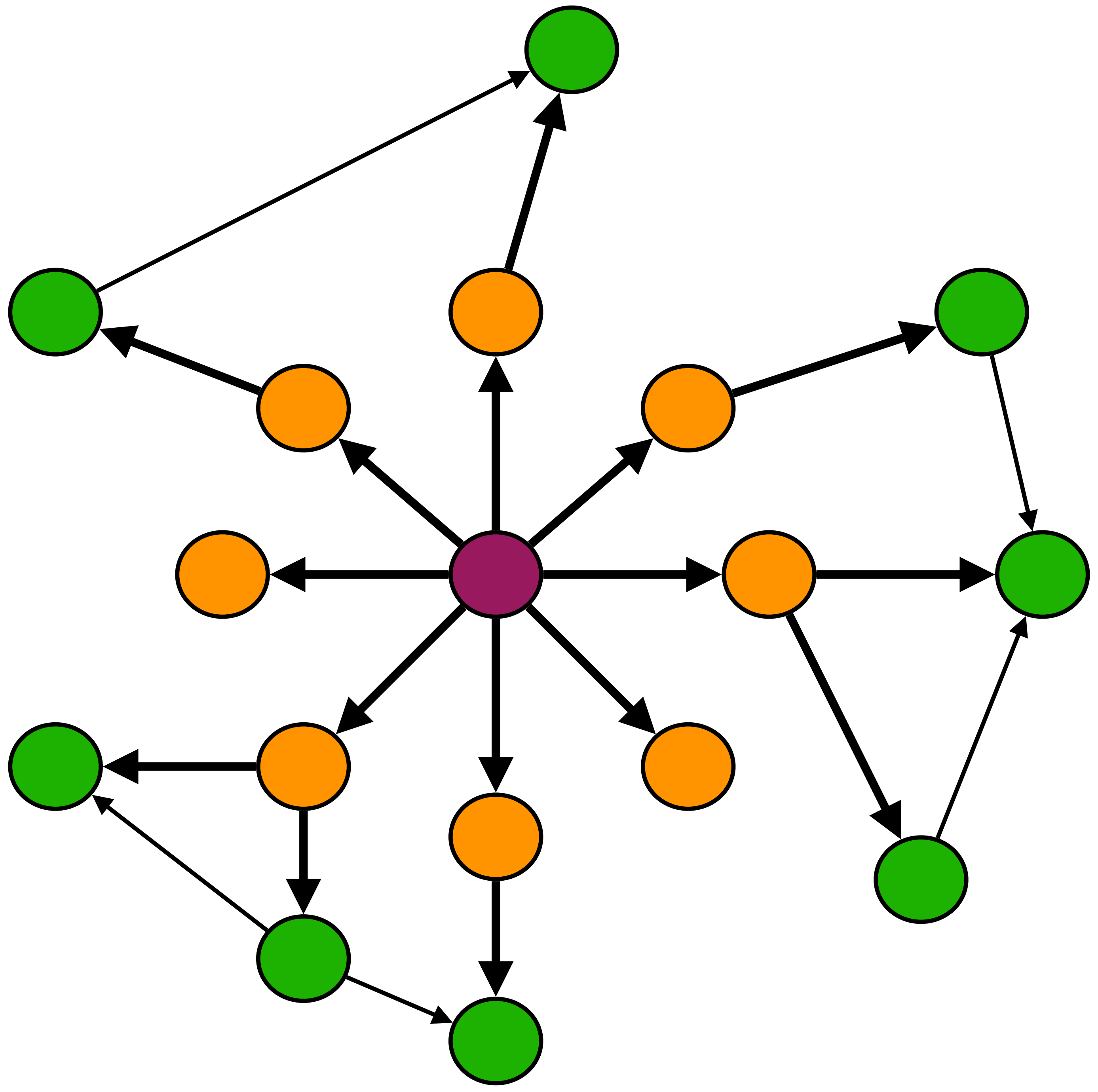


Data-Driven Algorithms



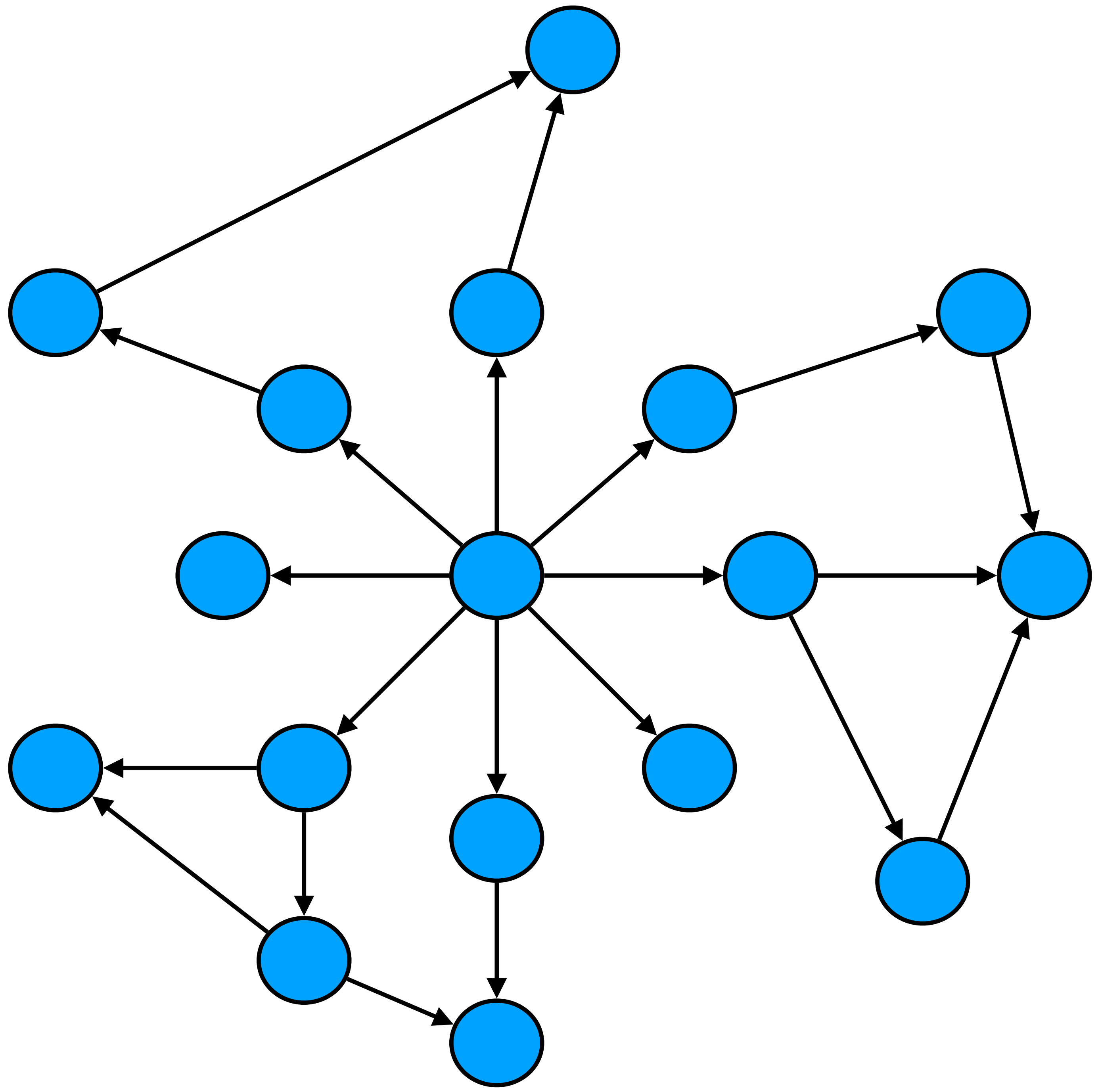
Work on a subset of vertices and edges
(Data-Driven)

Data-Driven Algorithms



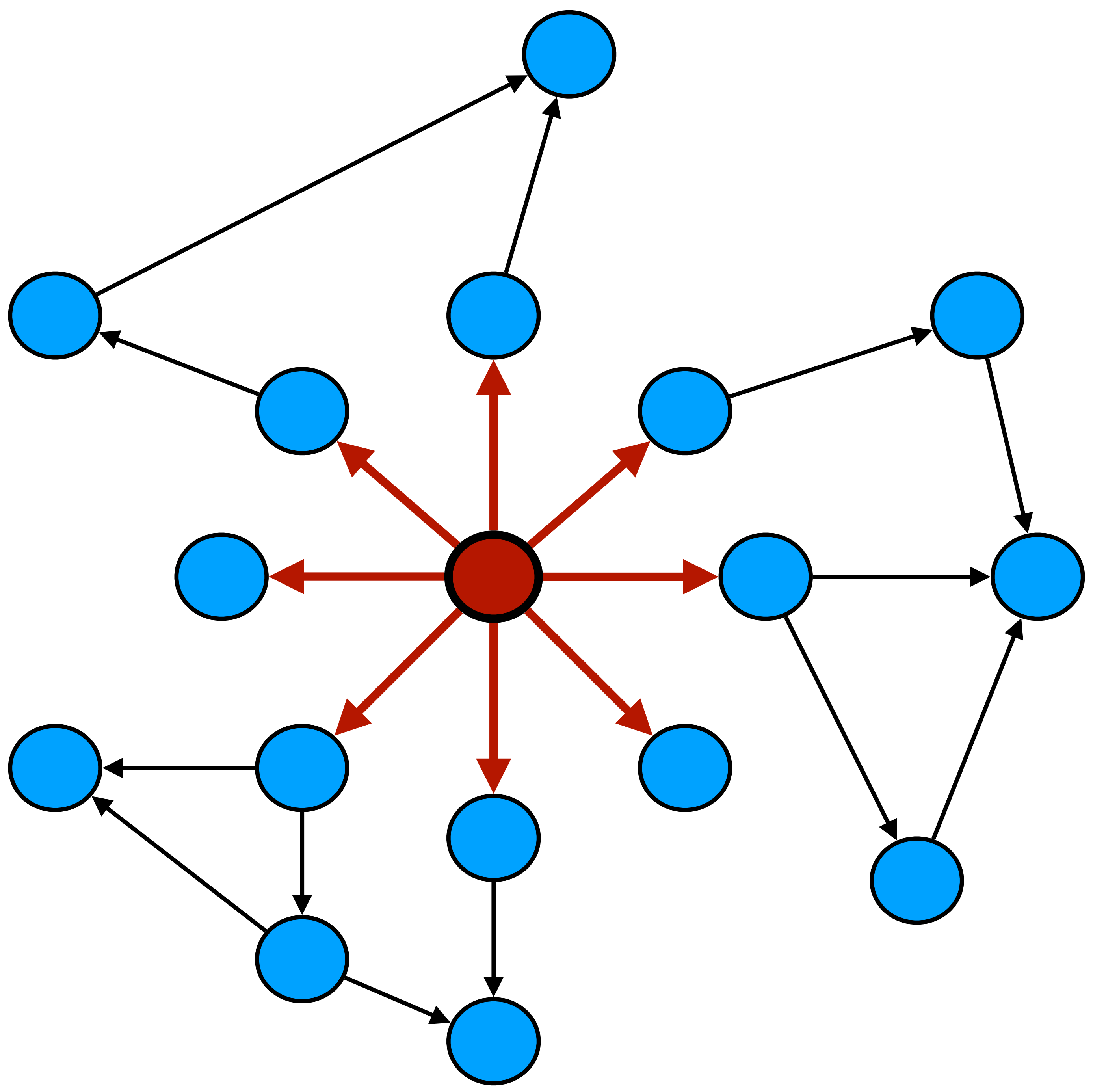
Work on a subset of vertices and edges
(Data-Driven)

Graph Traversal



- Different Traversal Orders have different performance characteristics

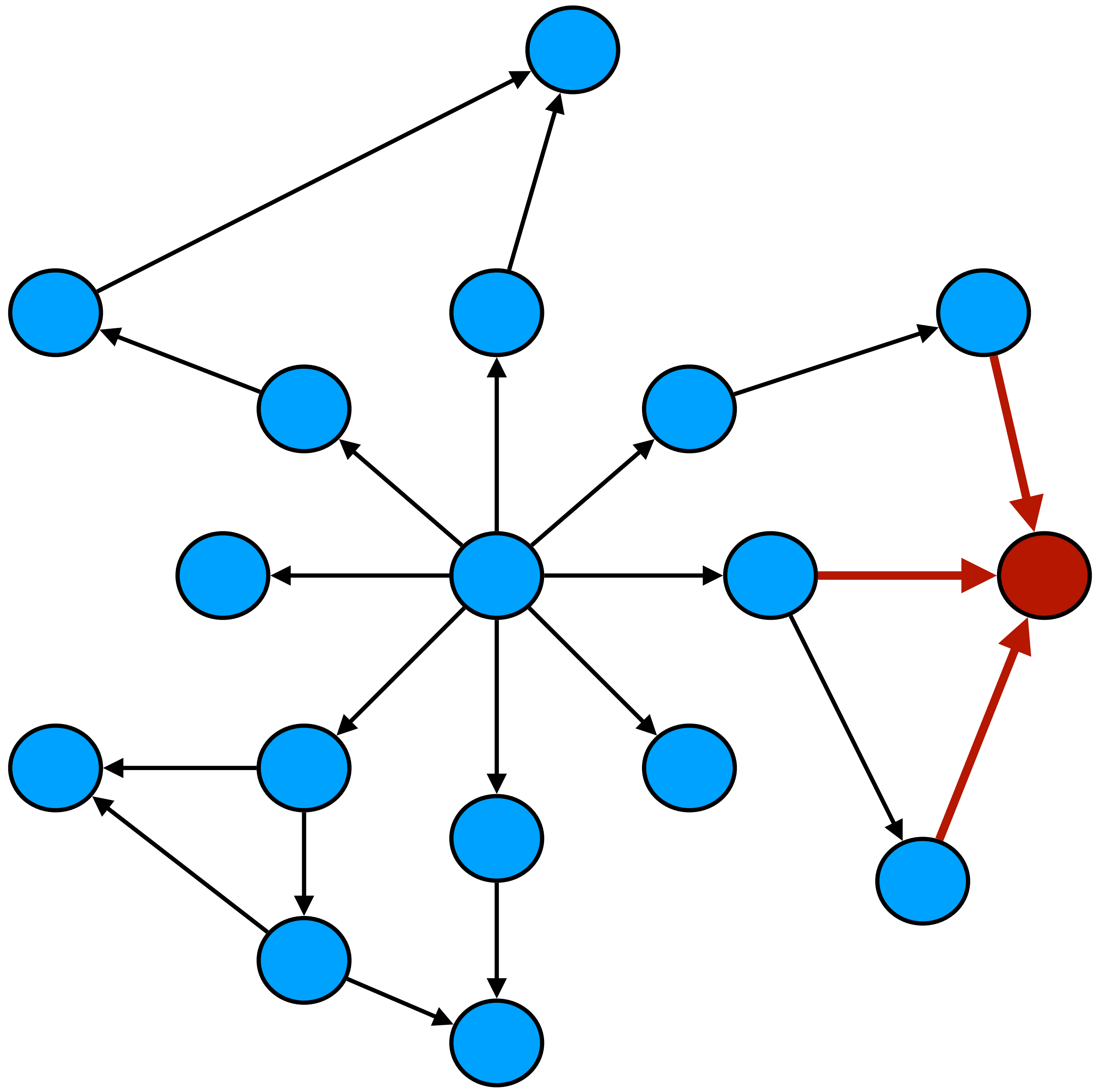
Push Traversal



Incurs overhead with atomics

Traverses no extra edges

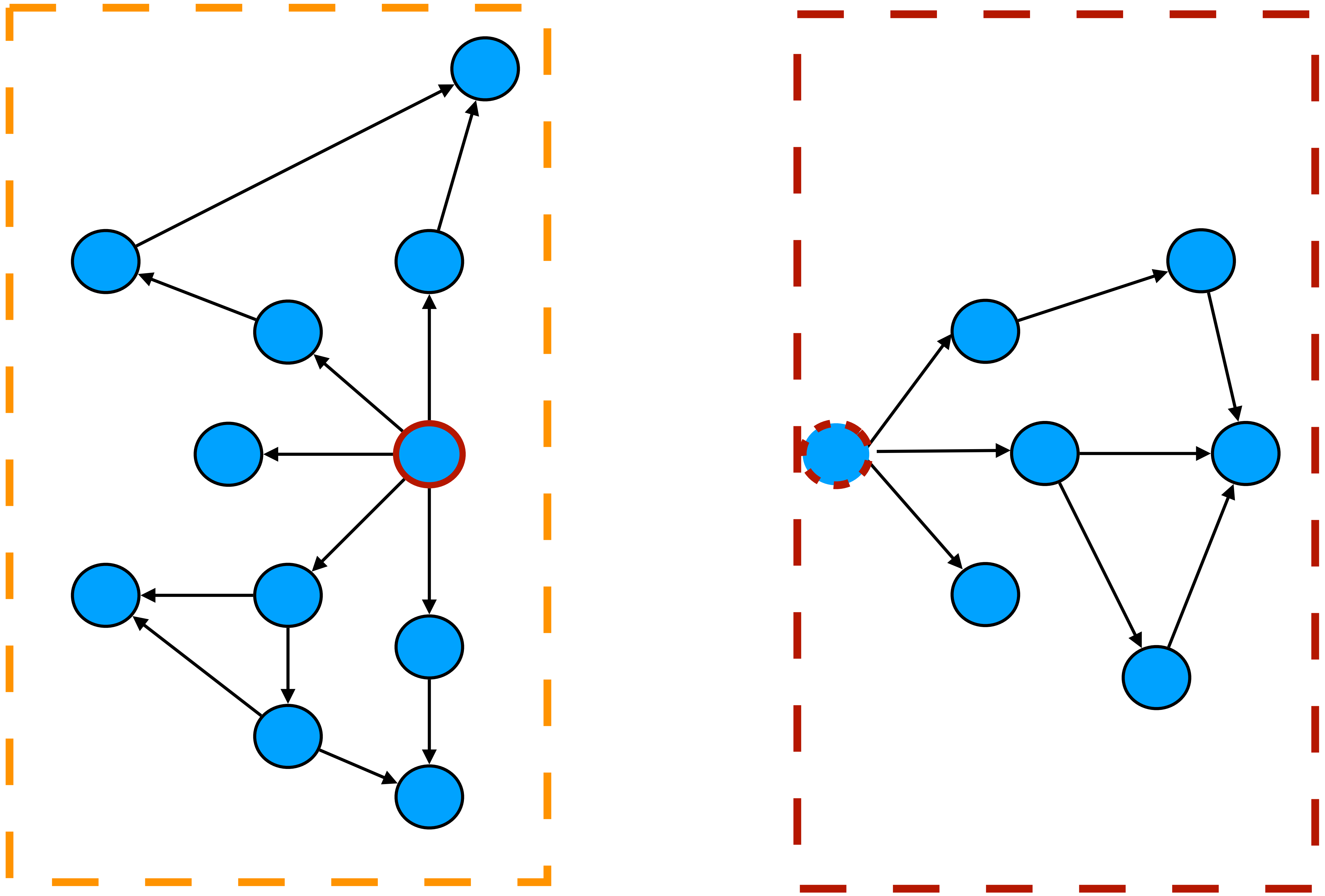
Pull Traversal



Incurs no overhead from atomics

Traverses extra edges

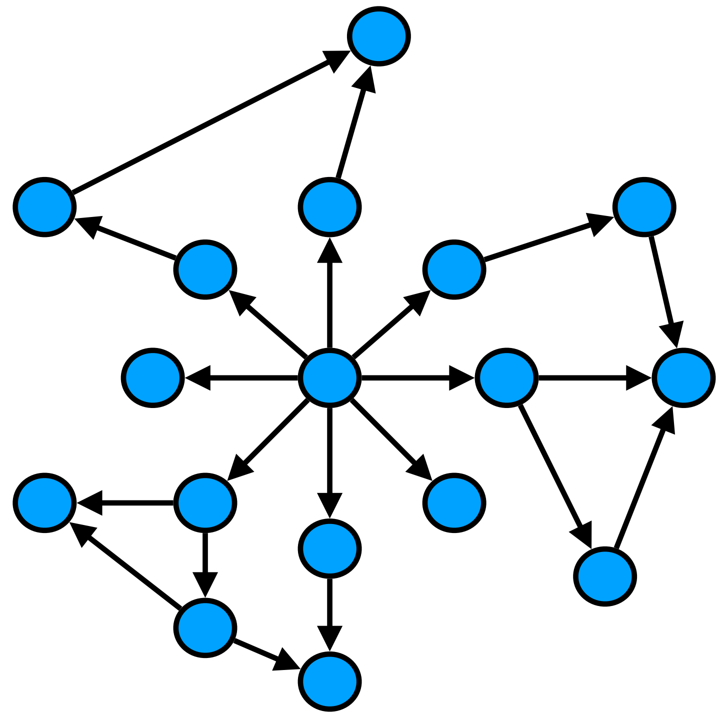
Partitioning



Improves locality

Needs extra instructions to traverse two graphs

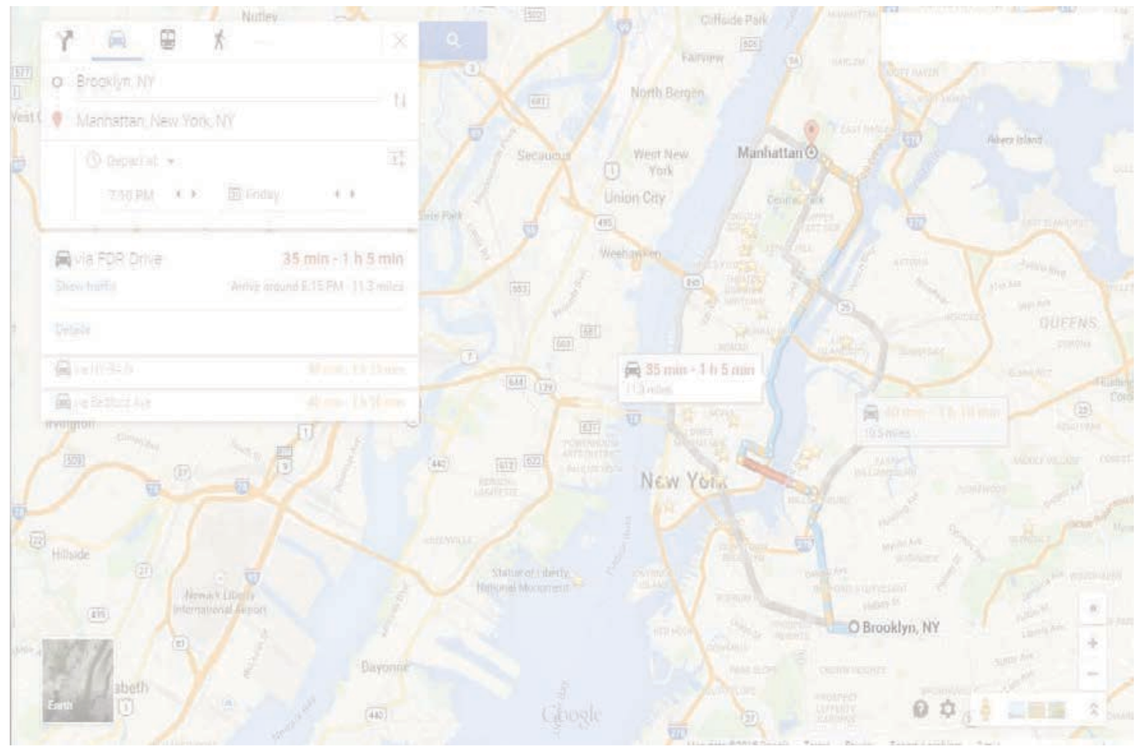
Power-Law Graphs



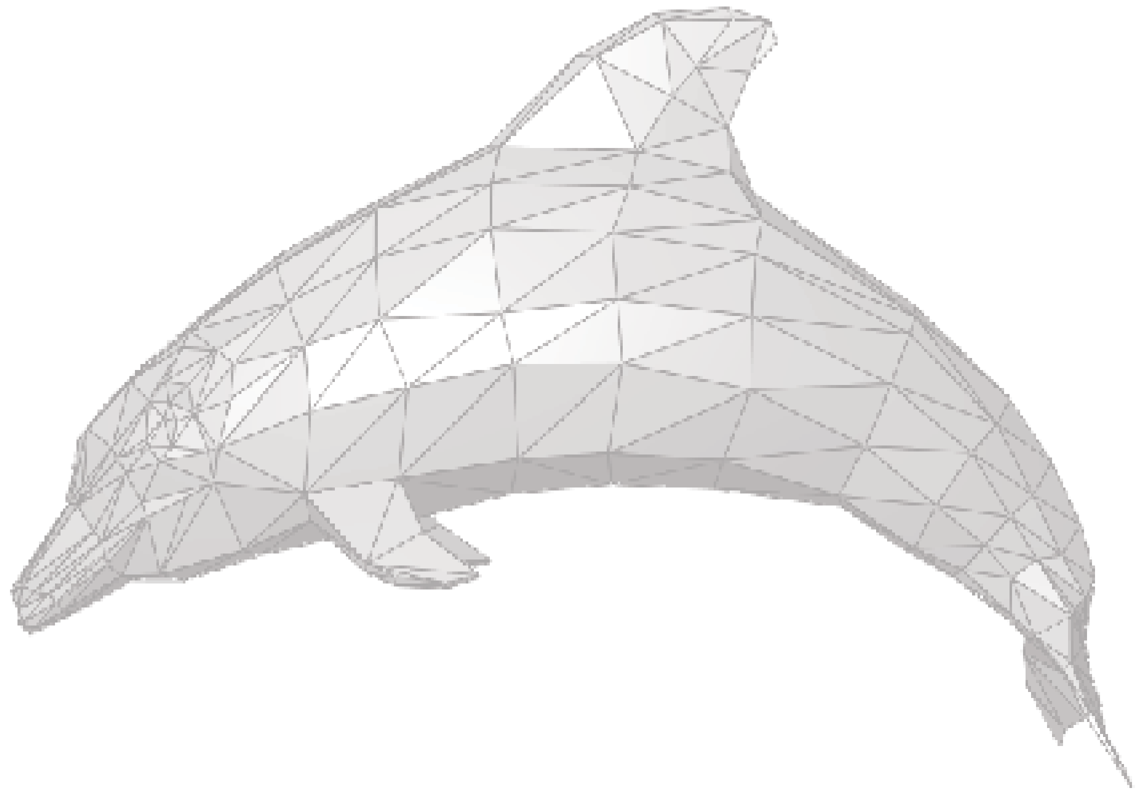
World Wide Web

Power-Law Degree Distribution, Small Diameter, Poor Locality

Social Networks

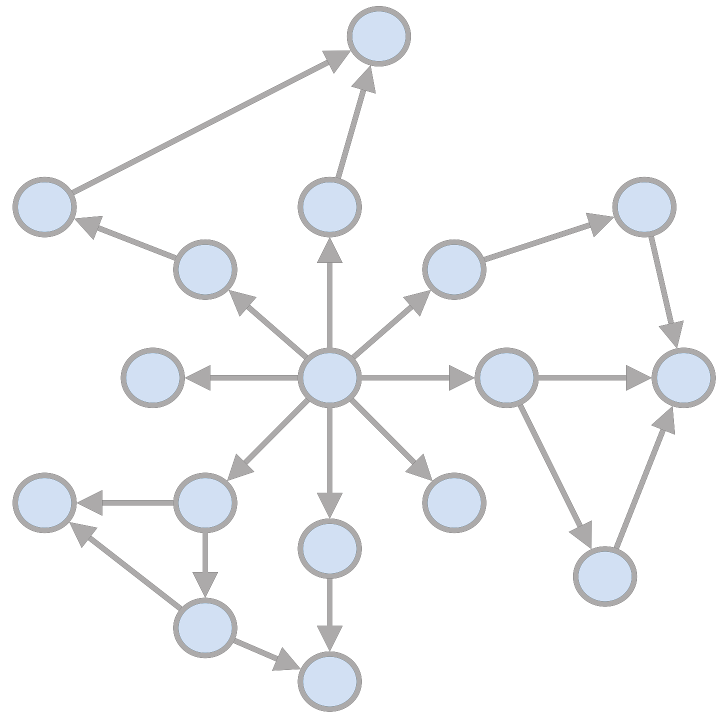


Maps



Engineering Meshes

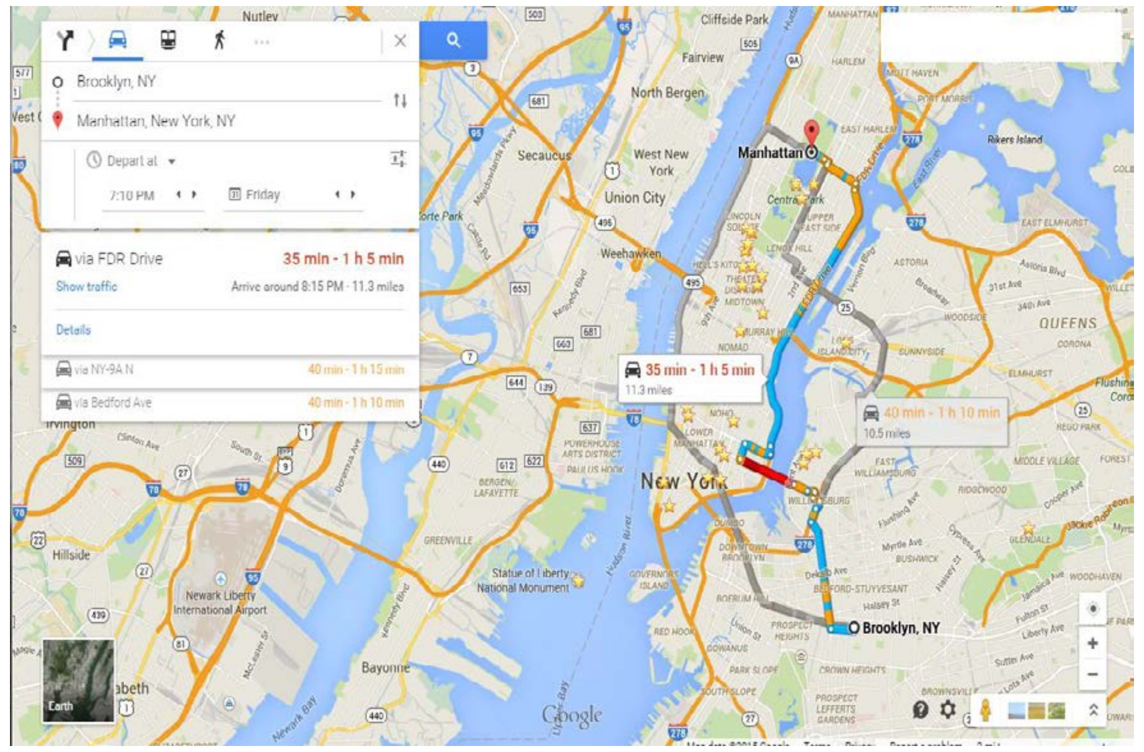
Bounded-Degree Graphs



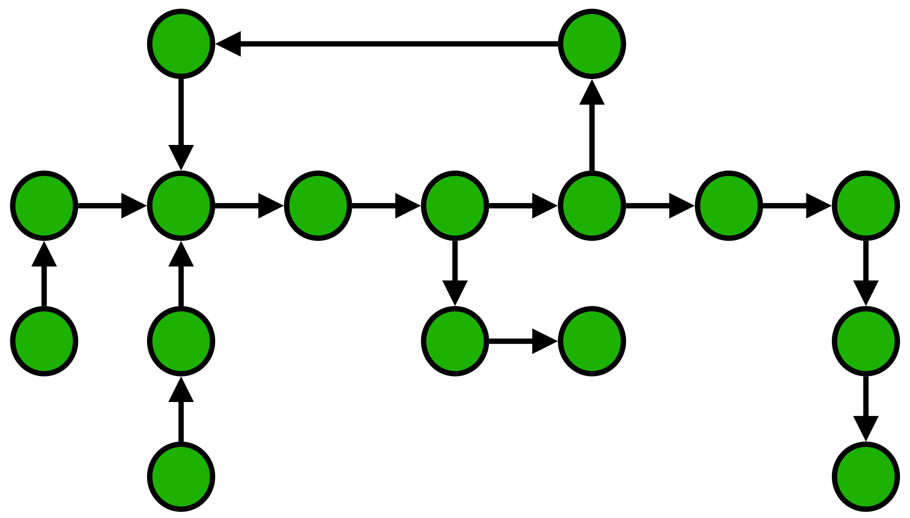
World Wide Web

Power-Law Degree Distribution,
Small Diameter, Poor Locality

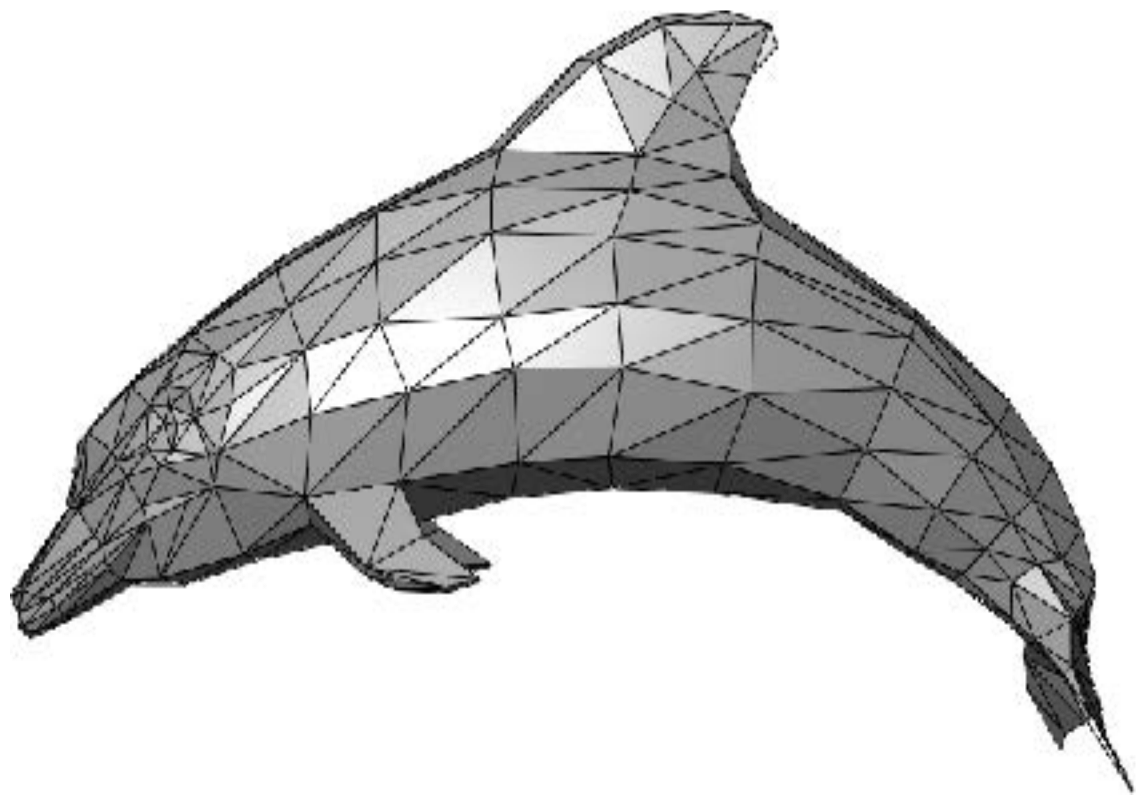
Social Networks



Maps

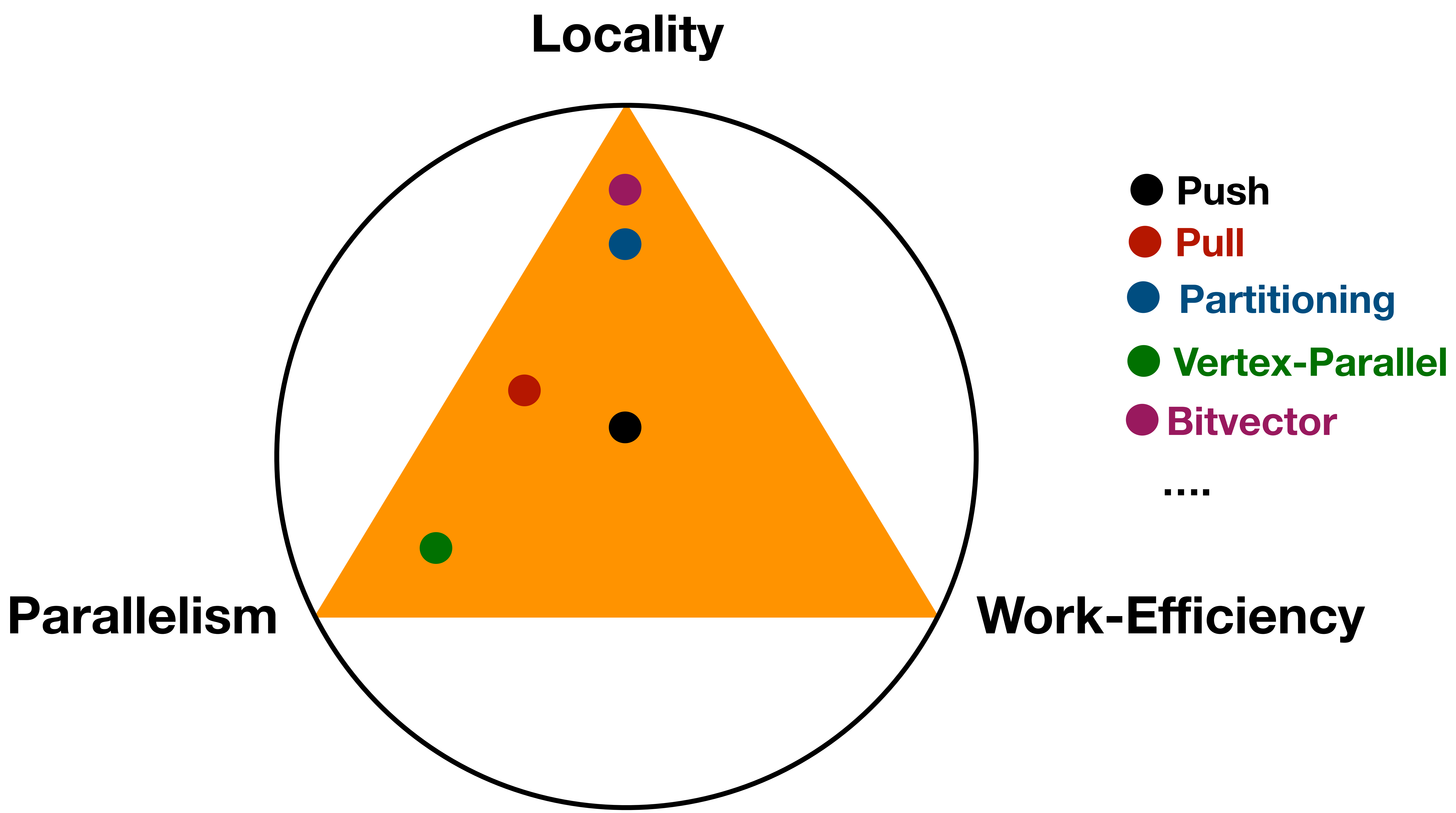


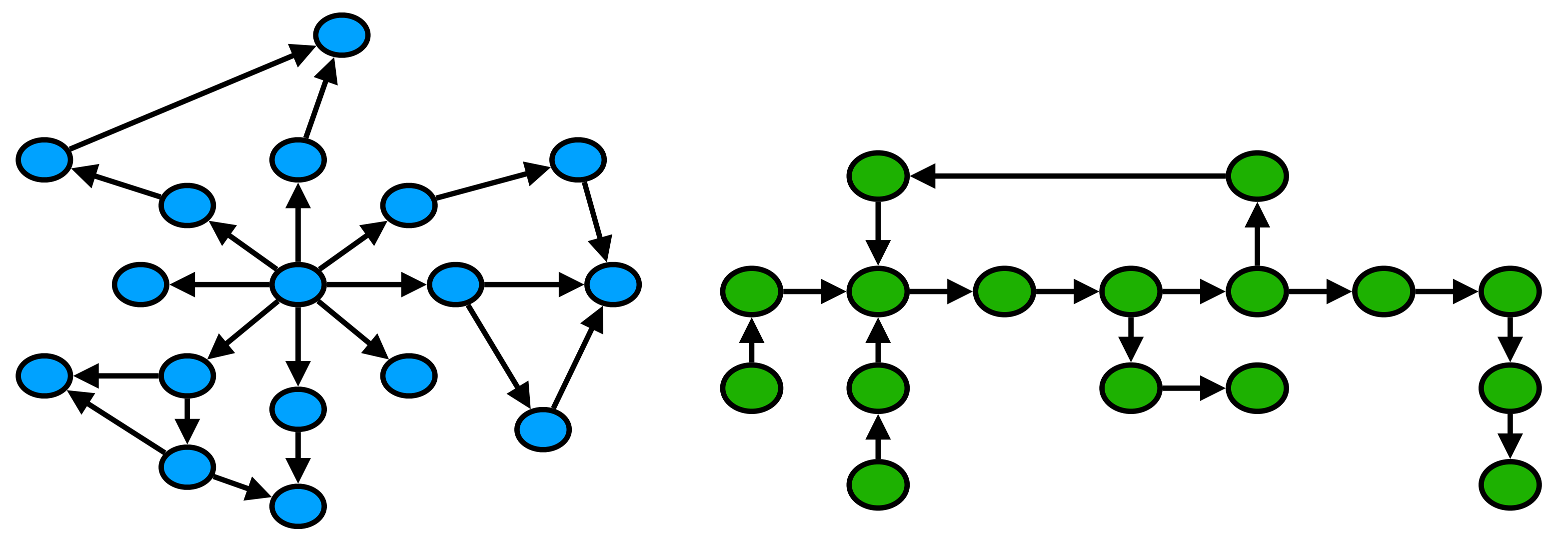
Bounded Degree Distribution
Large Diameter, Excellent Locality



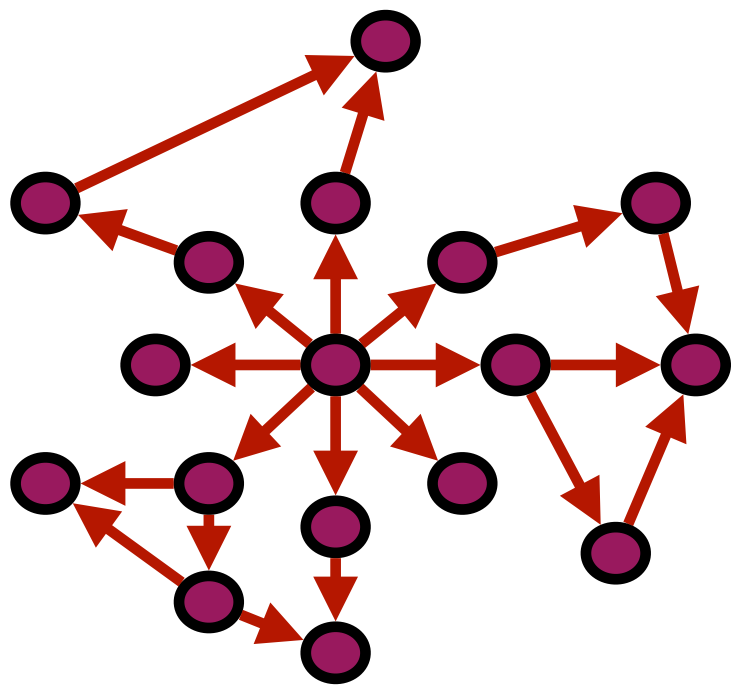
Engineering Meshes

Optimization Tradeoff Space

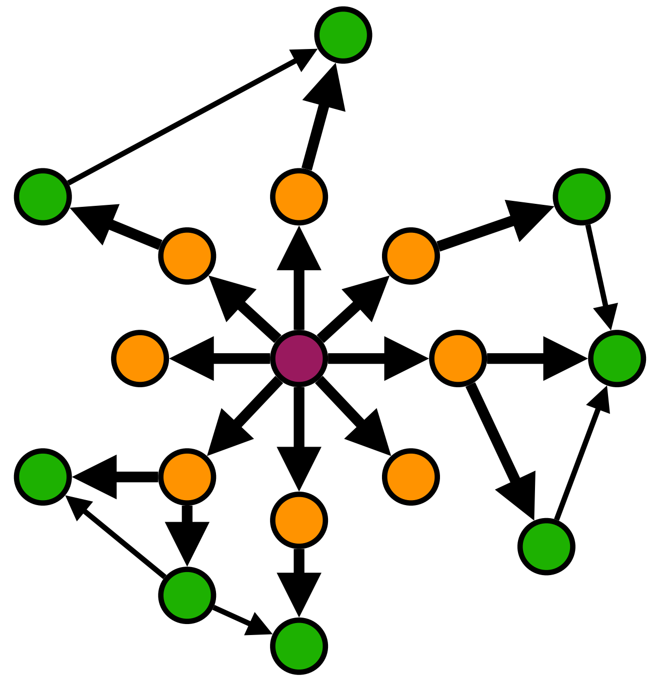


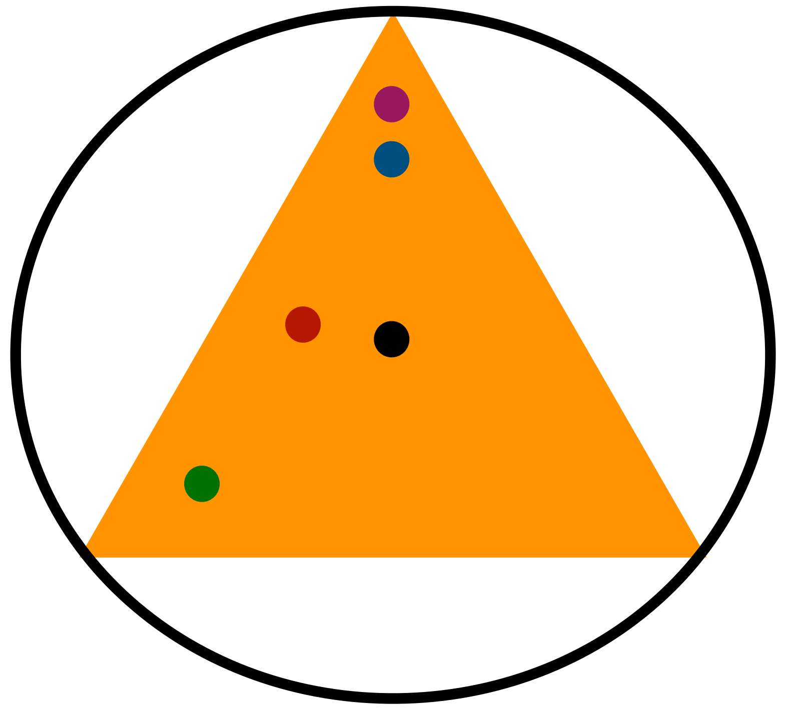


Graphs



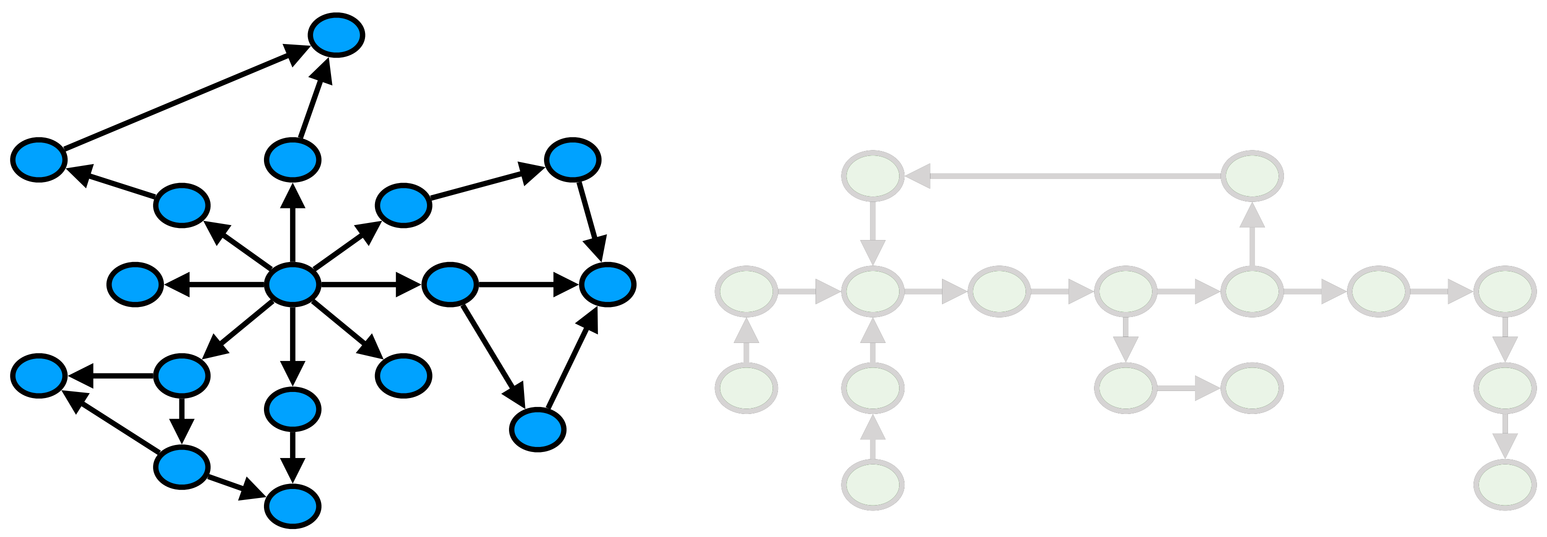
Algorithms



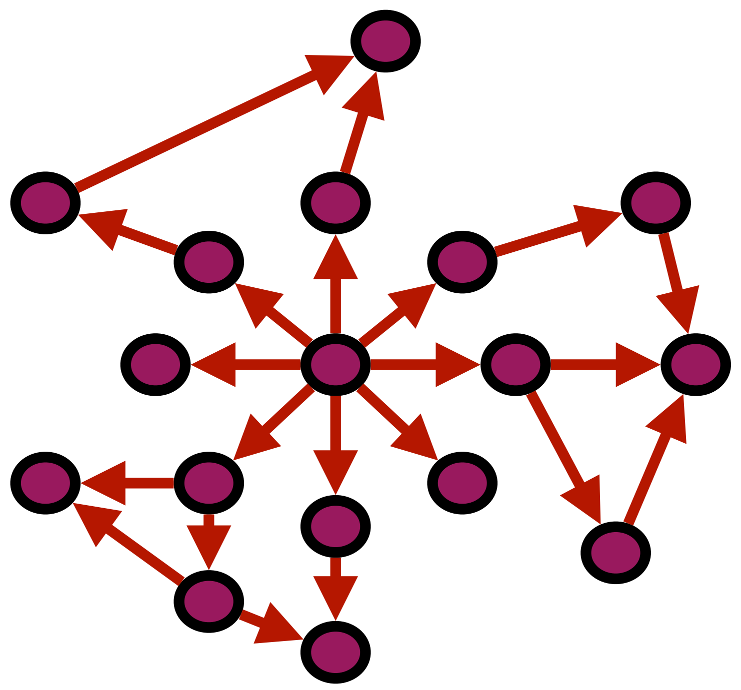


Optimizations

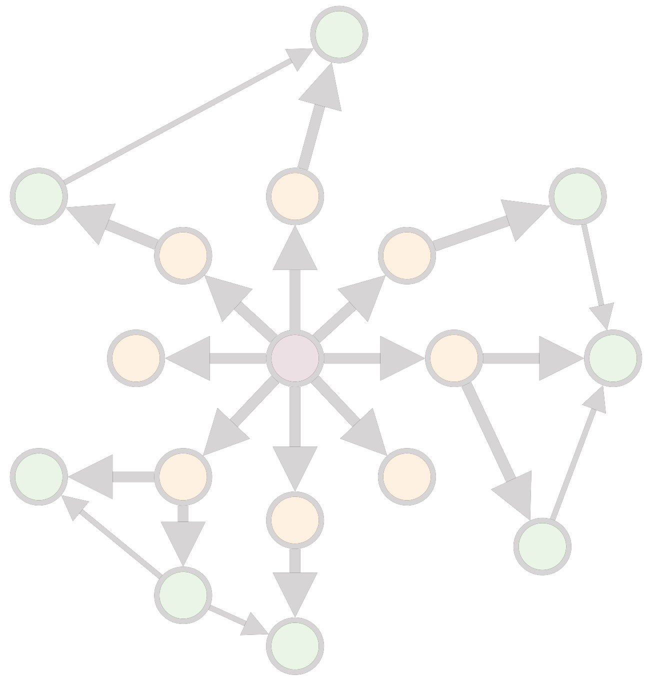
Hardware

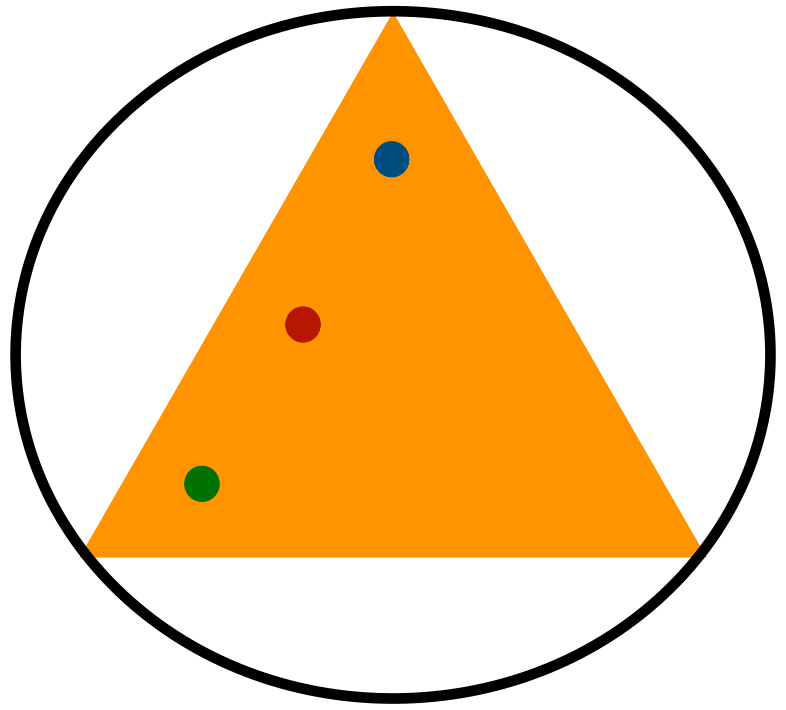


Graphs



Algorithms





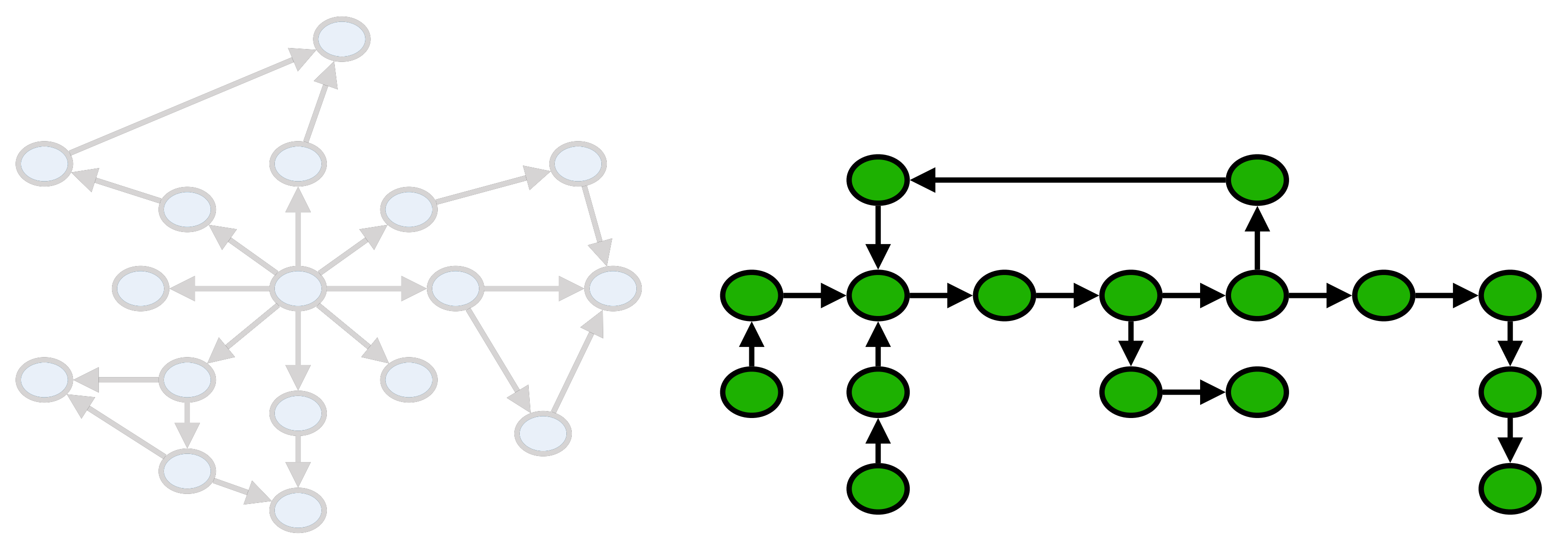
Optimizations

Hardware

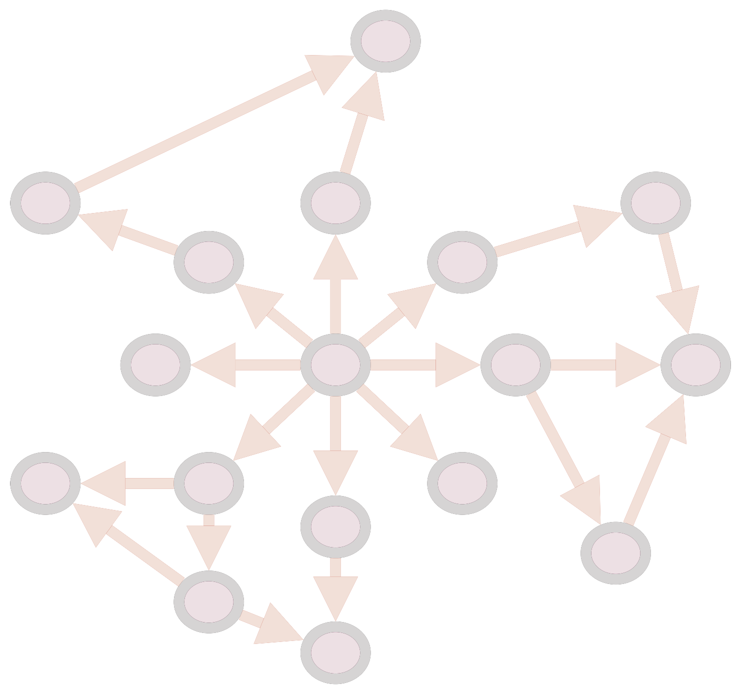
Pull

Partitioning

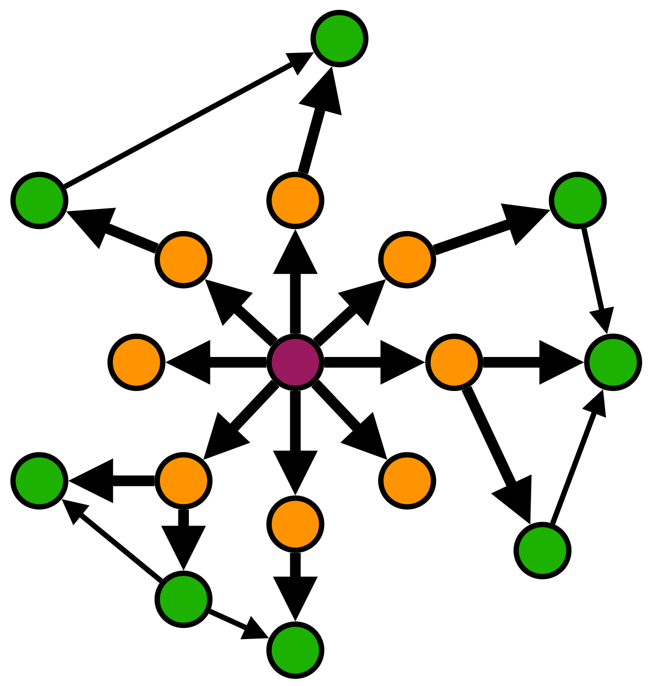
Vertex-Parallel

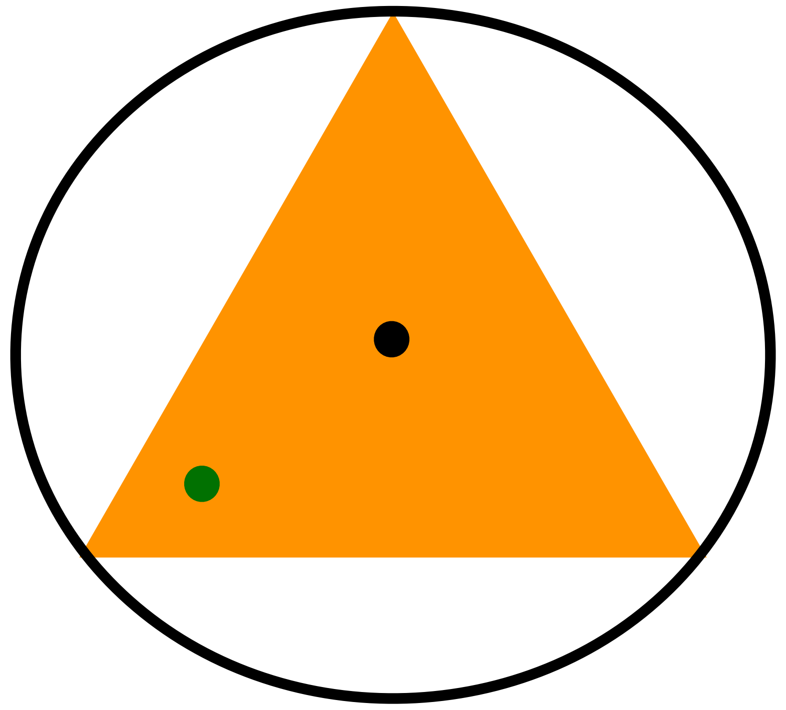


Graphs



Algorithms

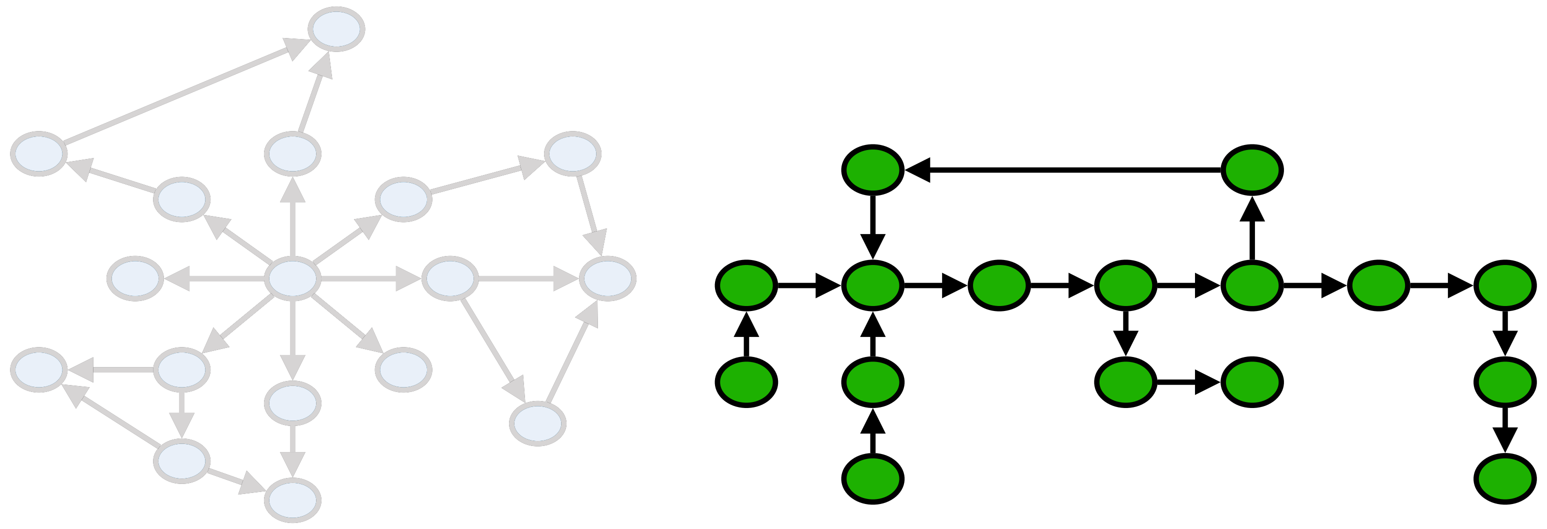




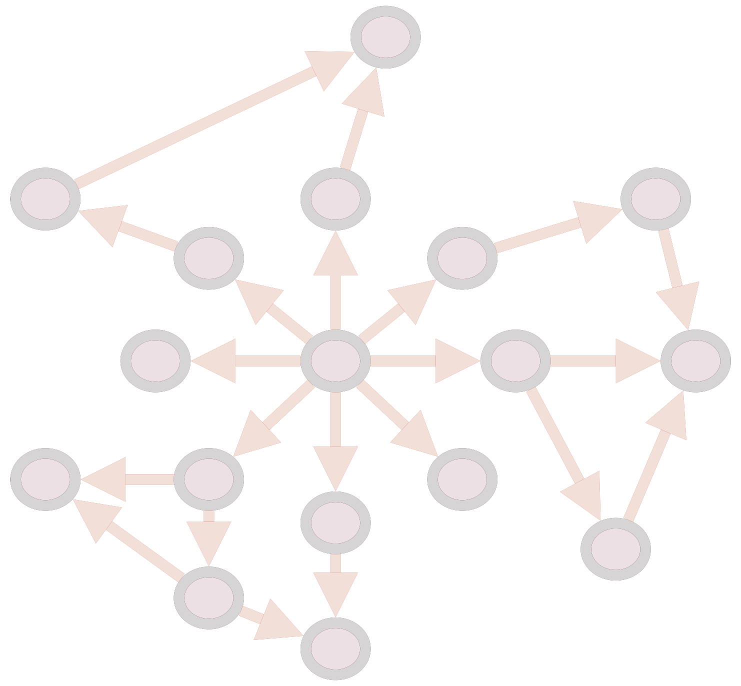
Optimizations

Hardware

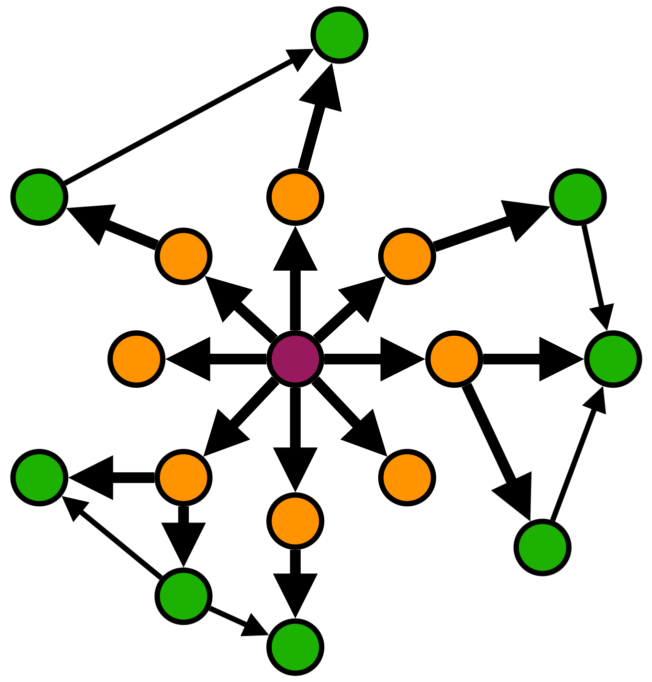
Push
Vertex-Parallel



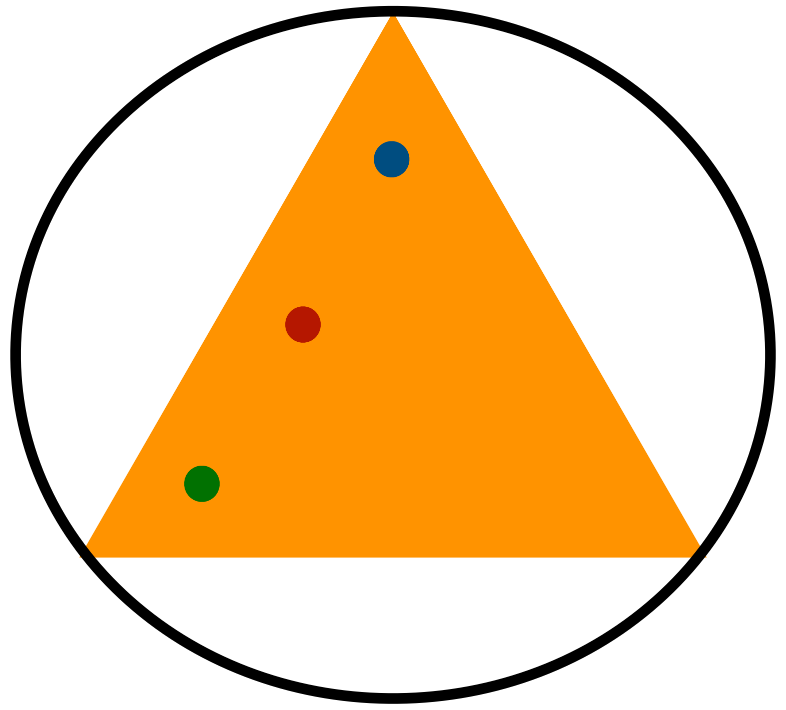
Graphs



Algorithms



Bad optimizations (schedules) can be > 100x slower



Optimizations

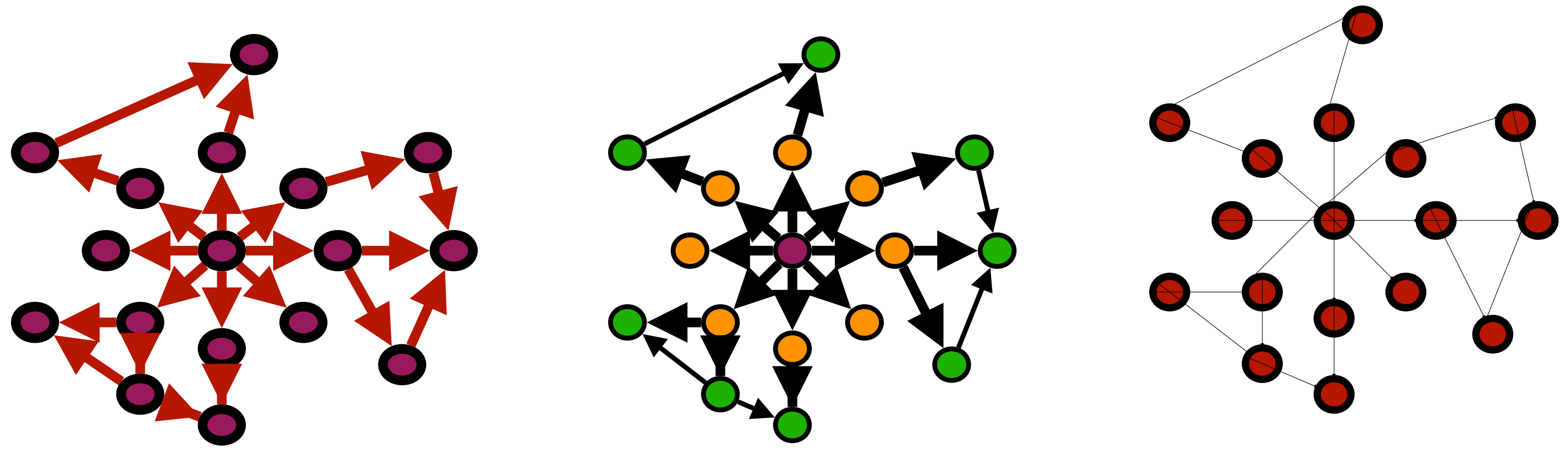
Pull
Partitioning
Vertex-Parallel

Hardware

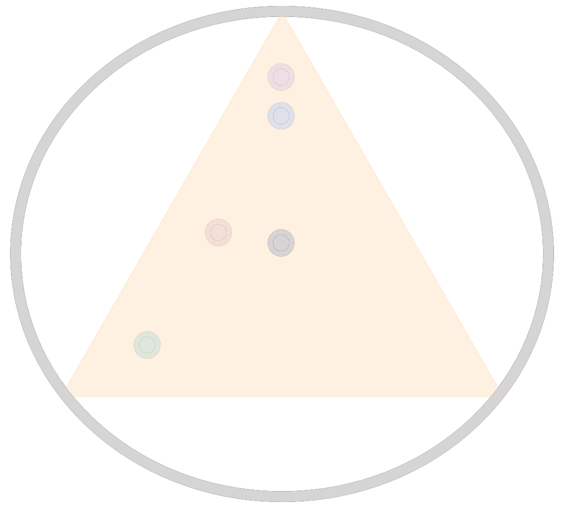
GraphIt

A Domain-Specific Language for Graph Applications

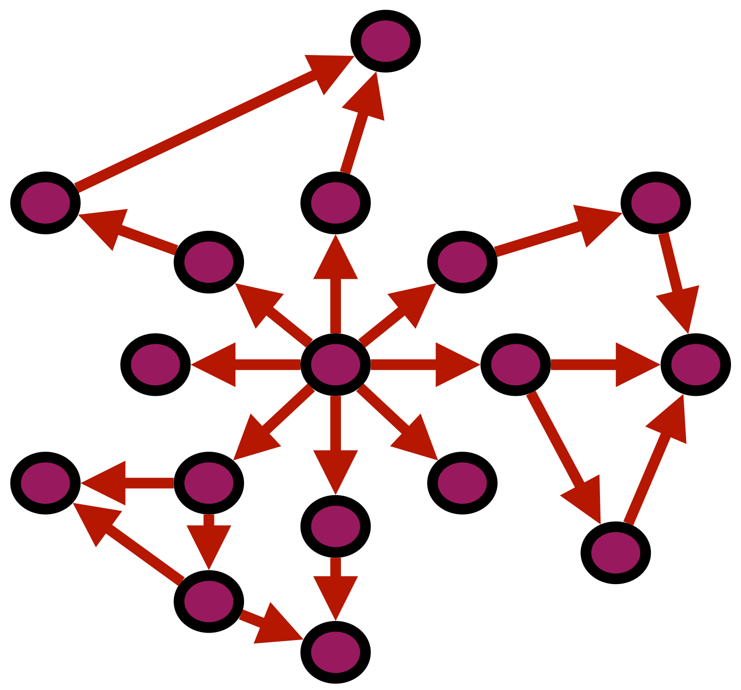
- **Decouple algorithm from optimization for graph applications**
- **Algorithm**: What to Compute
  - **High level** ignores all the optimization details



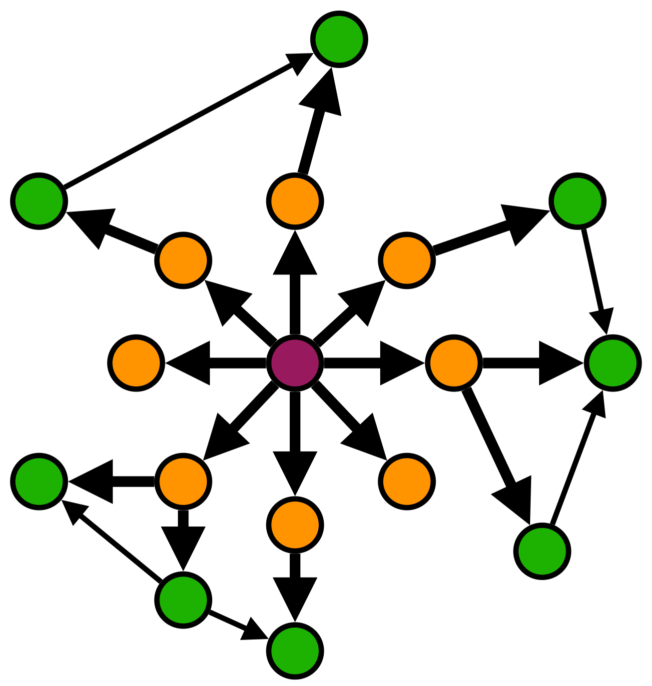
- **Optimization (schedule):** How to Compute
  - **Easy to use** for users to try different combinations
  - **Powerful** enough to beat hand-hand-optimized libraries by up to 4.8x



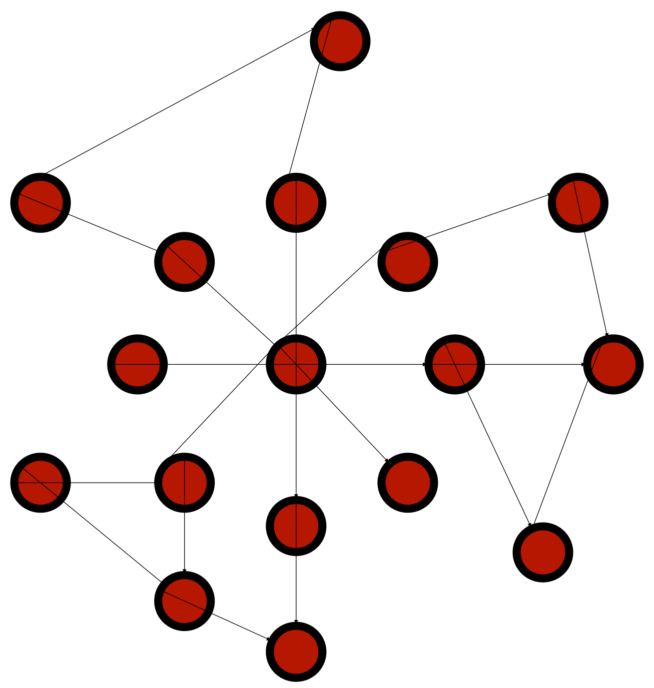
Algorithm Language



edges.apply(func)

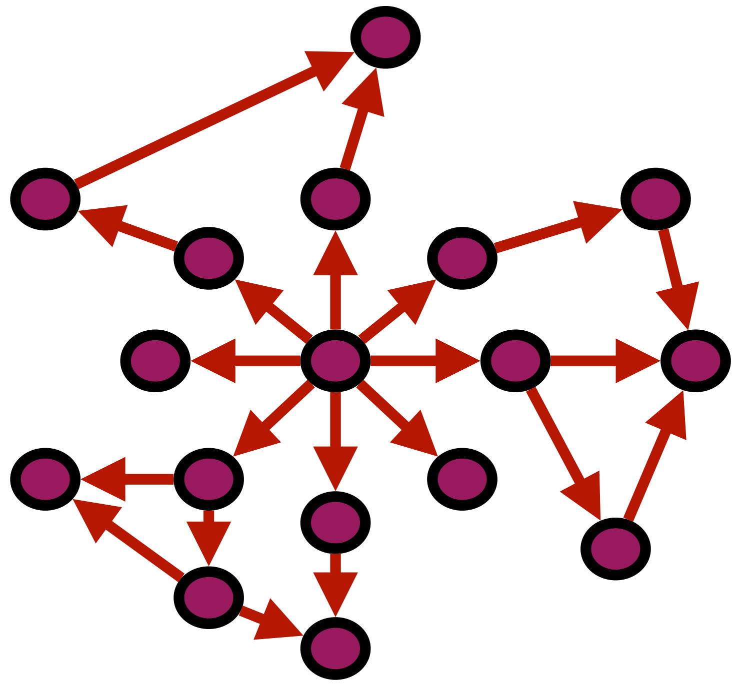


edges.from(vertexset)
.to(vertexset)
.srcFilter(func)
.dstFilter(func)
.apply(func)

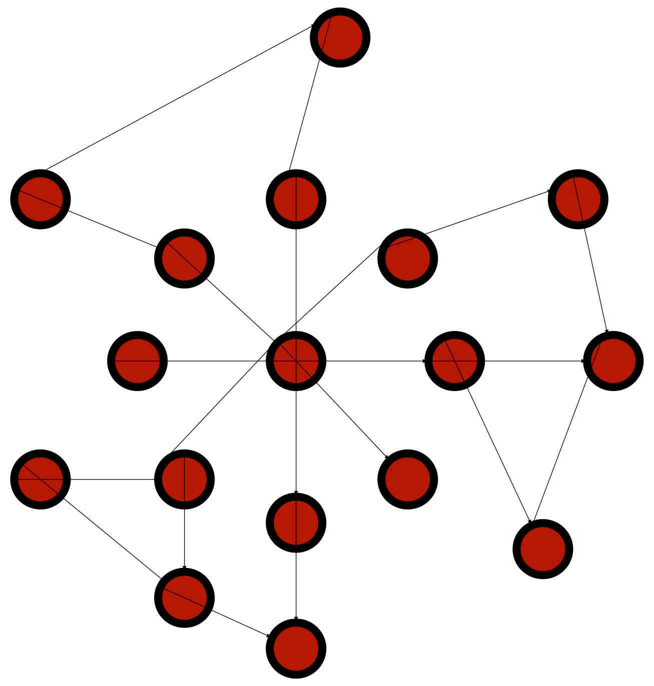


vertices.apply(func)

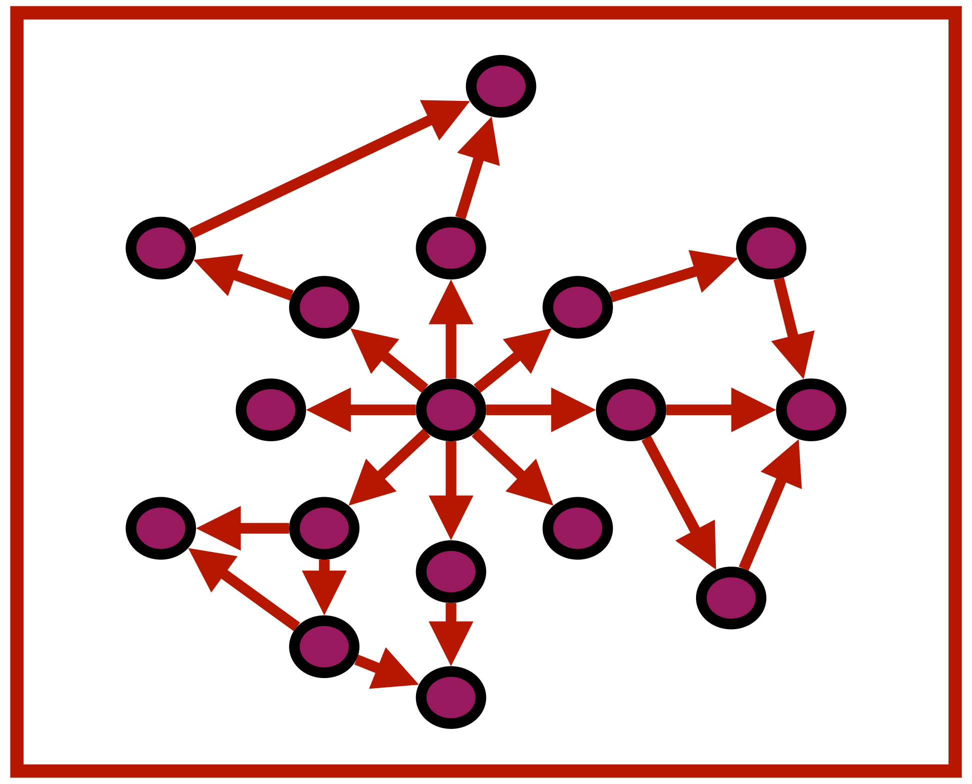
PageRank Example



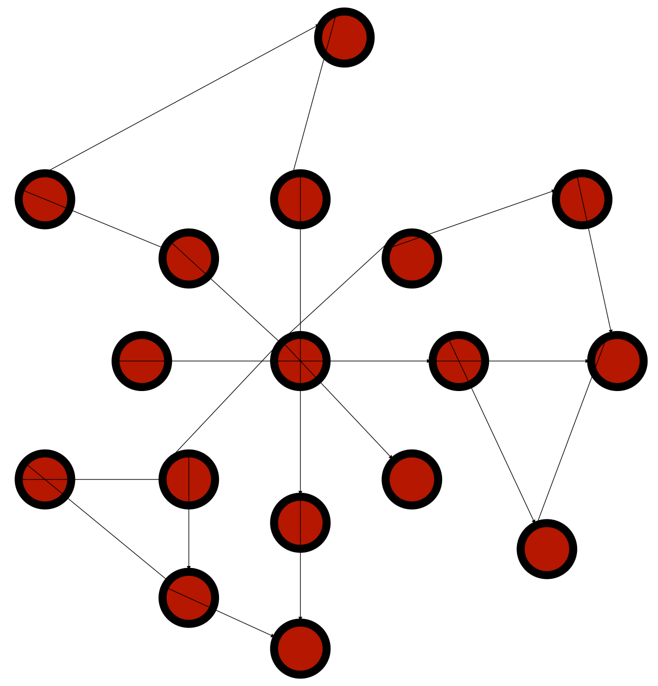
```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
     end
 end
```



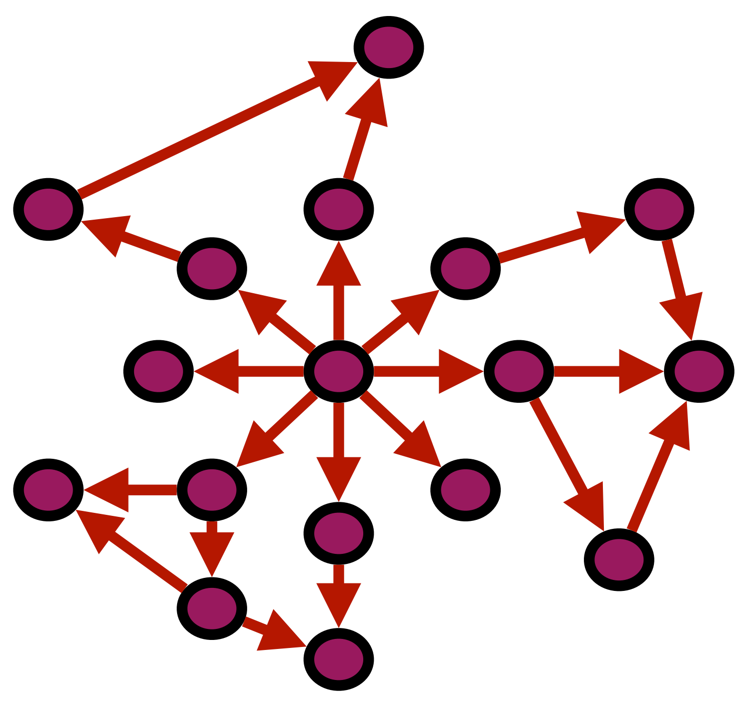
PageRank Example



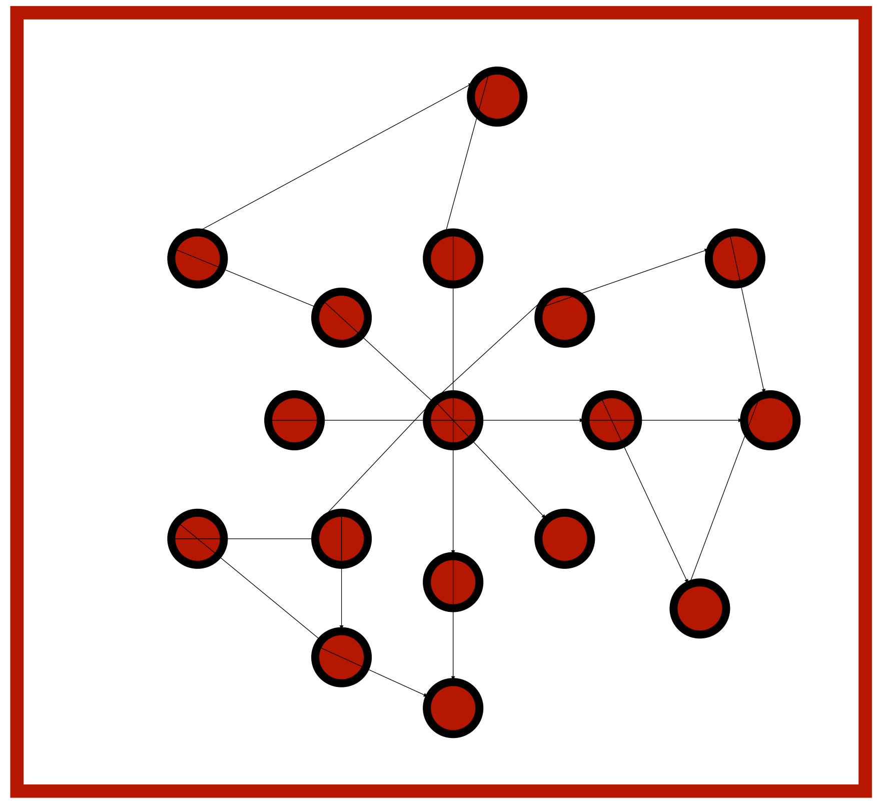
```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```



PageRank Example



```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end
```



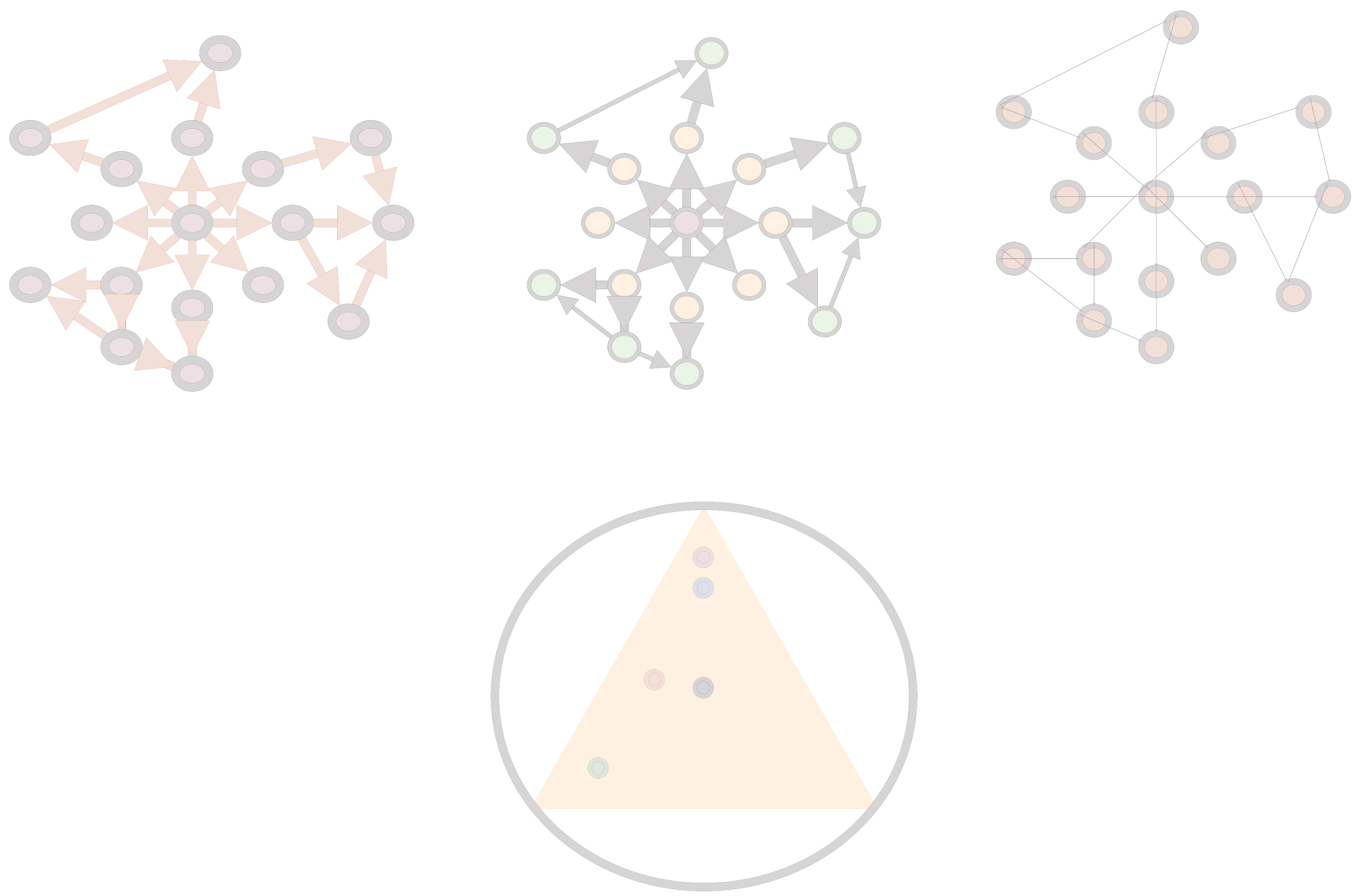
```
func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end
```

```
func main()
    for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
end
```

GraphIt

A Domain-Specific Language for Graph Applications

- **Decouple algorithm from optimization for graph applications**
- **Algorithm**: What to Compute
  - **High level** ignores all the optimization details



Scheduling Language

```
func updateEdge (src: Vertex, dst: Vertex)
    new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
    new_rank[v] = beta_score + 0.85*new_rank[v];
    old_rank[v] = new_rank[v];
    new_rank[v] = 0;
end

 func main()
     for i in 1:max_iter
        #s1# edges.apply(updateEdge);
        vertices.apply(updateVertex);
    end
 end
```

Scheduling Language

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

 func main()
    for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
    end
 end
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Scheduling Functions

```
schedule:
   program->configApplyDirection("s1", "SparsePush");
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Scheduling Functions

```
schedule:
   program->configApplyDirection("s1", "SparsePush");
```

Schedule 1

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     new_rank[dst]  +=  old_rank[src] / out_degree[src];
   }
}

….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "SparsePush");
```

Schedule 2

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID src : vertices) {
  for(NodeID dst : G.getOutNgh(src)){
     atomic_add (new_rank[dst],
                       old_rank[src] / out_degree[src] );
  }
}

….
```

Scheduling Functions

```
schedule:
   program->configApplyDirection("s1", "SparsePush");
   program->configApplyParallelization("s1", "dynamic-vertex-parallel");
```

Schedule 3

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…

parallel_for (NodeID dst : vertices) {
  for(NodeID src : G.getInNgh(dst)){
     new_rank[dst]  += old_rank[src] / out_degree[src];
   }
}

….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection(“s1”, “DensePull”);
    program->configApplyParallelization(“s1”, “dynamic-vertex-parallel”);
```

Schedule 4

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:max_iter
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

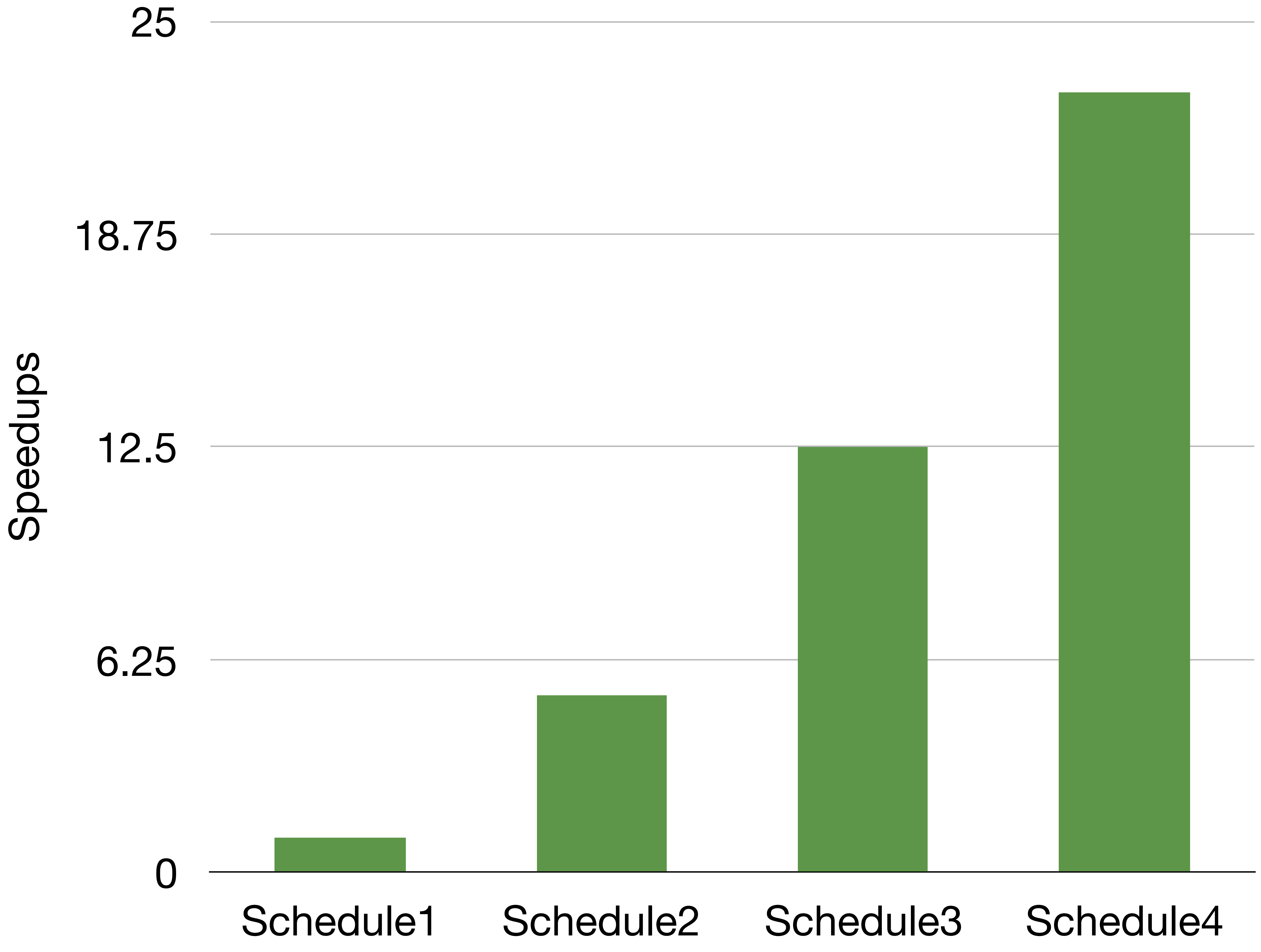
Pseudo Generated Code

```
double * new_rank = new double[num_verts];
double * old_rank = new double[num_verts];
int * out_degree = new int[num_verts];

…
for (Subgraph sg : G.subgraphs) {
  parallel_for (NodeID dst : verticesa) {
    for(NodeID src : G.getInNgh(dst)){
     new_rank[dst]  += old_rank[src] / out_degree[src];
    }
  }
}
….
```

Scheduling Functions

```
schedule:
    program->configApplyDirection("s1", "DensePull");
    program->configApplyParallelization("s1", "dynamic-vertex-parallel");
    program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Speedups of Schedules



Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Many More Optimizations

- Direction optimizations (configApplyDirection),
  - SparsePush, DensePush, DensePull, DensePull-SparsePush, DensePush-SparsePush
- Parallelization strategies (configApplyParallelization)
  - serial, dynamic-vertex-parallel, static-vertexparallel, edge-aware-dynamic-vertex-parallel, edge-parallel
- Cache (configApplyNumSSG)
  - fixed-vertex-count, edge-aware-vertexcount
- NUMA (configApplyNUMA)
  - serial, static-parallel, dynamic-parallel
- AoS, SoA (fuseFields)
- Vertexset data layout (configApplyDenseVertexSet)
  - bitvector, boolean array

State of the Art and GraphIt

	PR	BFS	CC	SSSP
LJ	3.48	1	1	1
TW	5.63	1.13	3.12	1.14
WB	4.15	1.42	2.96	1.13
RD	2.69	4.81	2.16	4.57
FT	6.17	1.38	4.94	2.77

Ligra
(PPoPP13)

	PR	BFS	CC	SSSP
LJ	1.64	3.7	5.98	1.86
TW	2.34	9.4	11	1.62
WB	2.14	7.44	9.13	2.98
RD	1.61	9.06	7.04	151
FT				

GraphMat
(VLDB15)

	PR	BFS	CC	SSSP
LJ	1.51	1.83	3.06	1.82
TW	2.42	6.03	5.78	1.41
WB	2.59	2.84	5.96	2.54
RD	1.26	2.45	8.99	328
FT				

GreenMarl
(ASPLOS12)

	PR	BFS	CC	SSSP
LJ	8.15	1.41	2.05	1.78
TW	3.53	4.49	5.68	1.43
WB	2.82	1.83	8.07	1.36
RD	13	1.02	1.05	3.25
FT	3.61	7.02	7.05	1.08

Galois
(SOSP13)

	PR	BFS	CC	SSSP
LJ	1.26	2.22	2.46	1.57
TW	1.26	1.64	4.33	1
WB	1	1.52	4.93	1.67
RD	1.49	48.8	7.08	26.1
FT	1.37	1.49	5.24	1.43

Gemini
(OSDI16)

	PR	BFS	CC	SSSP
LJ	1.08	1.93	1.38	
TW	1.8	1.17	1.94	
WB	1.26	1.28	1.64	
RD	1	8.26	1	
FT	1.67	1.04	2.24	

Grazelle
(PPoPP18)

	PR	BFS	CC	SSSP
LJ	1	1.3	1.11	1.07
TW	1	1	1	1
WB	1	1	1	1
RD	1.23	1	1.43	1
FT	1	1	1	1

GraphIt
(OOPSLA18)

Intel Xeon E5-2695 v3 CPUs with 12 cores each for a total of 24 cores and 48 hyper-threads.

Halide

- A new language & compiler
  - Originally developed for image processing
  - Focuses on stencils on regular girds
  - Complex pipelines of stencil kernels
  - Support other operations like reductions and scans

- Primary goal
  - Match or exceed hand optimized performance on each architecture
  - Reduce the rote programming burden of highly optimized code
  - Increase the portability without loss of performance

A Simple Example: 3X3 Blur

```
void box_filter_3x3(const Image &in, Image &blury) {
  Image blurx(in.width(), in.height());  // allocate blurx array

      for (int y = 0; y < in.height(); y++)
        for (int x = 0; x < in.height(); x++)
    blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

      for (int x = 0; x < in.width(); x++)
        for (int y = 0; y < in.height(); y++)
    blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
```

Hand-Optimized C++

```
void box_filter_3x3(const Image &in, Image &blury) {
 __m128i one_third = _mm_set1_epi16(21846);
 #pragma omp parallel for
 for (int yTile = 0; yTile < in.height(); yTile += 32) {
  __m128i a, b, c, sum, avg;
  __m128i blurx[(256/8)*(32+2)]; // allocate tile blurx array
  for (int xTile = 0; xTile < in.width(); xTile += 256) {
   __m128i *blurxPtr = blurx;
   for (int y = -1; y < 32+1; y++) {
    const uint16_t *inPtr = &(in[yTile+y][xTile]);
    for (int x = 0; x < 256; x += 8) {
     a = _mm_loadu_si128((__m128i*)(inPtr-1));
     b = _mm_loadu_si128((__m128i*)(inPtr+1));
     c = _mm_load_si128((__m128i*)(inPtr));
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(blurxPtr++, avg);
     inPtr += 8;
   }}
   blurxPtr = blurx;
   for (int y = 0; y < 32; y++) {
    __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_load_si128(blurxPtr+(2*256)/8);
     b = _mm_load_si128(blurxPtr+256/8);
     c = _mm_load_si128(blurxPtr++);
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(outPtr++, avg);
}}}}}
```

9.9 → 0.9 ms/megapixel

11x faster *(quad core x86)*

Tiled, fused

Vectorized

Multithreaded

Redundant computation

Near roof-line optimum

Local Laplacian Filters

prototype for Adobe Photoshop Camera Raw / Lightroom

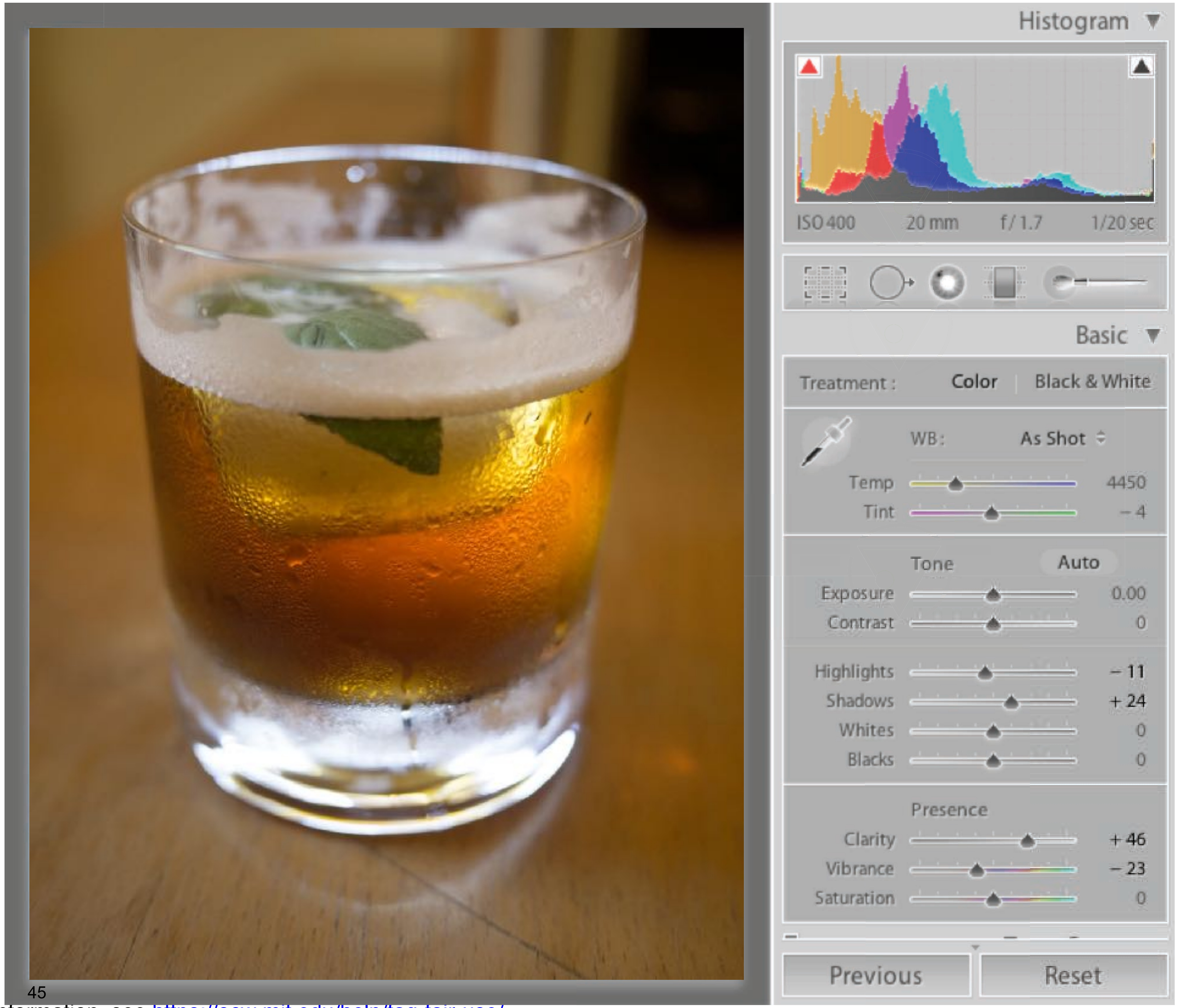
Reference: 300 lines C++

Adobe: 1500 lines
3 months of work
10x faster (vs. reference)

Halide: 60 lines
1 intern-day

20x faster (vs. reference)
2x faster (vs. Adobe)

GPU: 90x faster (vs. reference)
9x faster (vs. Adobe)



Decouple Algorithm From Schedule

- **Algorithm**: *what* is computed
  - The algorithm defines pipelines as pure functions
  - Pipeline stages are functions from coordinates to values
  - Execution order and storage are unspecified

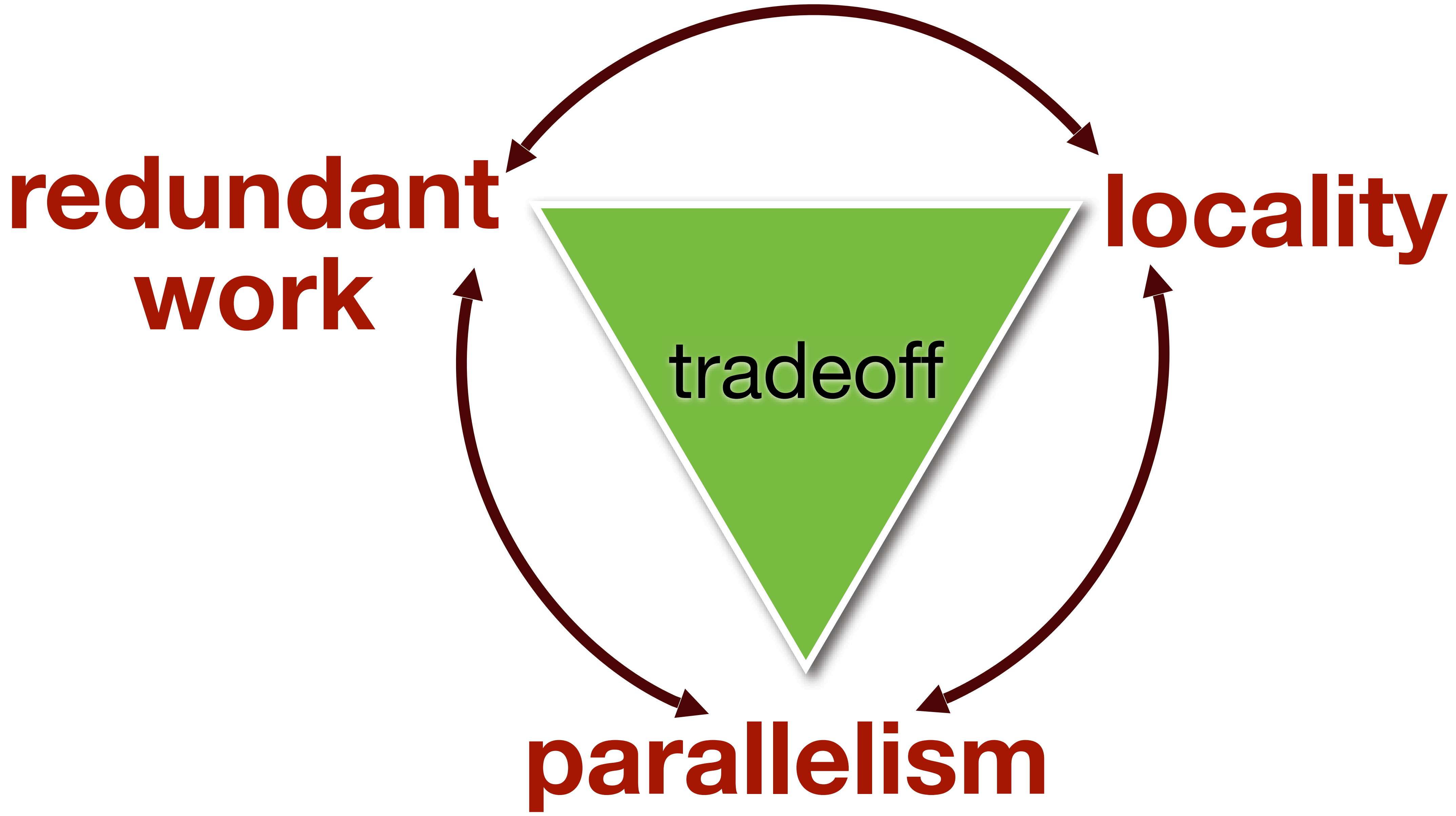
```
blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
```

Decouple Algorithm From Schedule

- **Algorithm**: *what* is computed
  - The algorithm defines pipelines as pure functions
  - Pipeline stages are functions from coordinates to values
  - Execution order and storage are unspecified

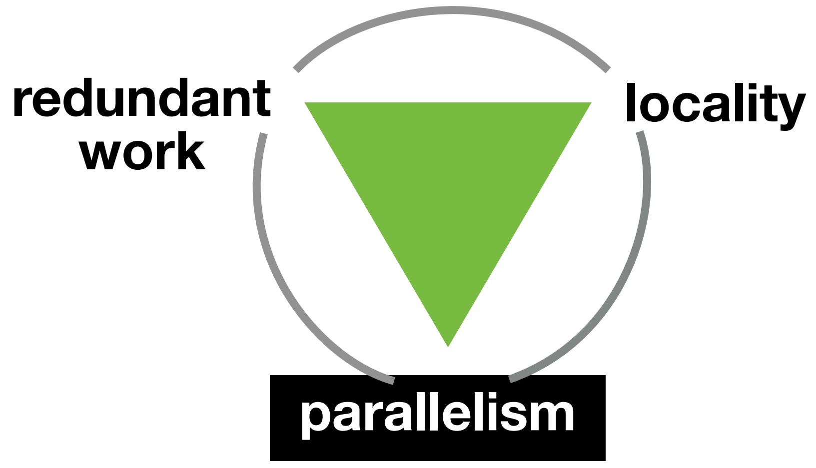
- **Schedule:** *where* and *when* it's computed
  - Architecture Specific
  - Single, unified model for all schedules
  - Simple enough to search, expose to user
    Powerful enough to beat expert-tuned code

Stencil Pipelines Require Tradeoffs Determined By Organization Of Computation

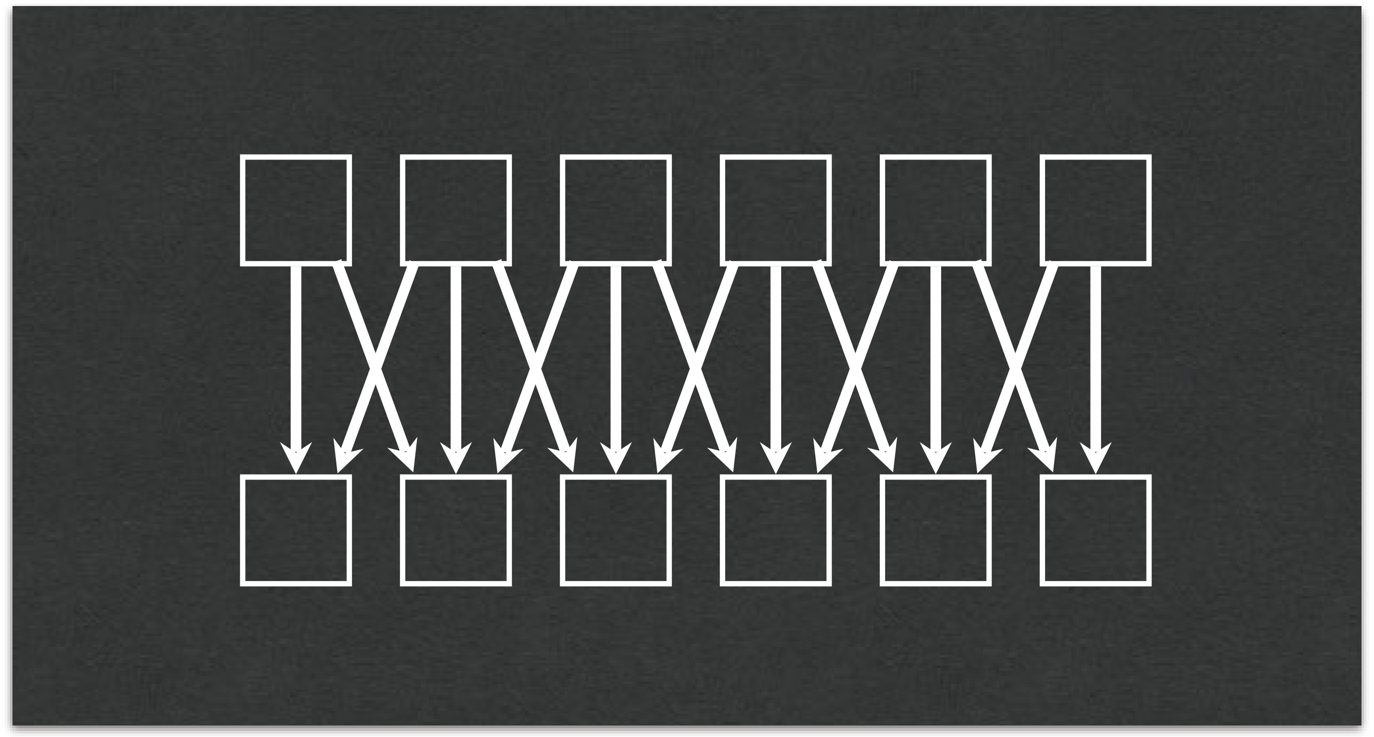


Parallelism

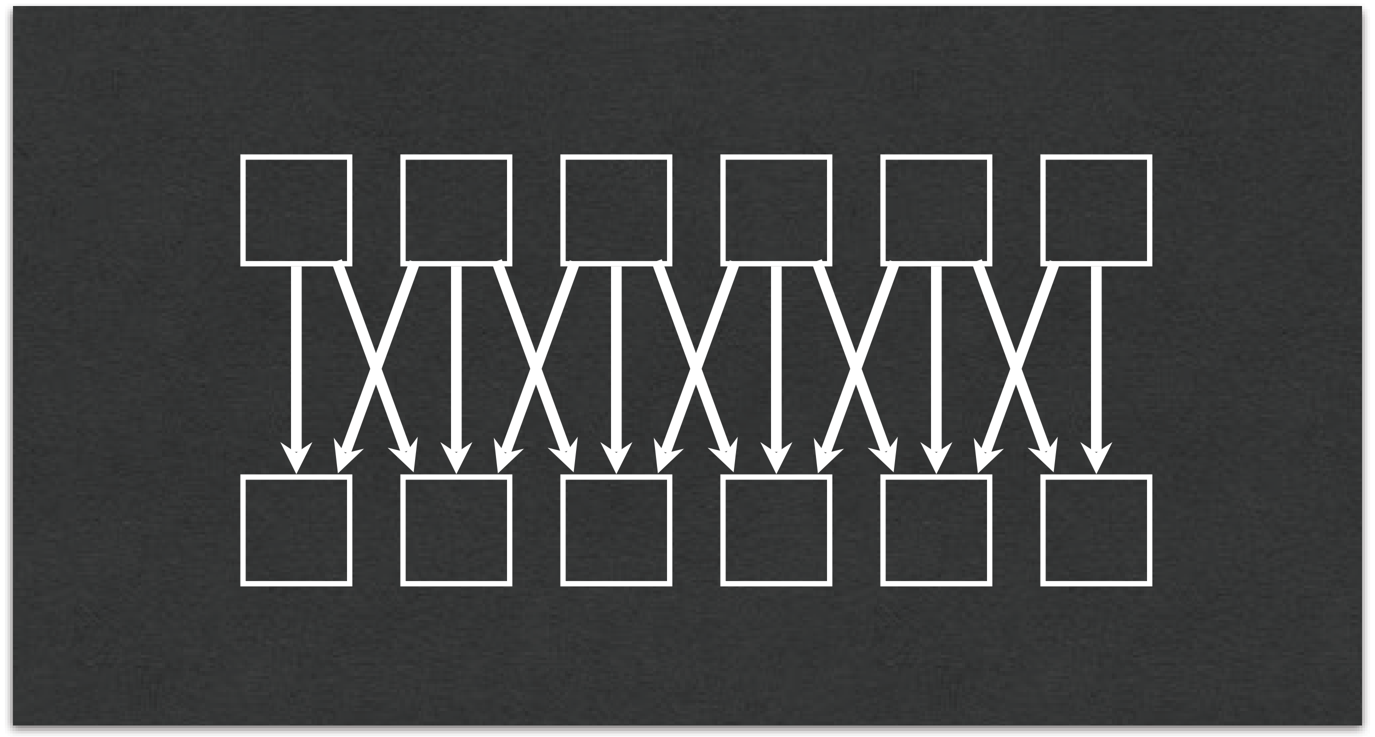
- **Need parallelism to keep multicores, vector units, clusters and GPUs busy**
  - **Too much parallelism is at best useless but can even be detrimental**



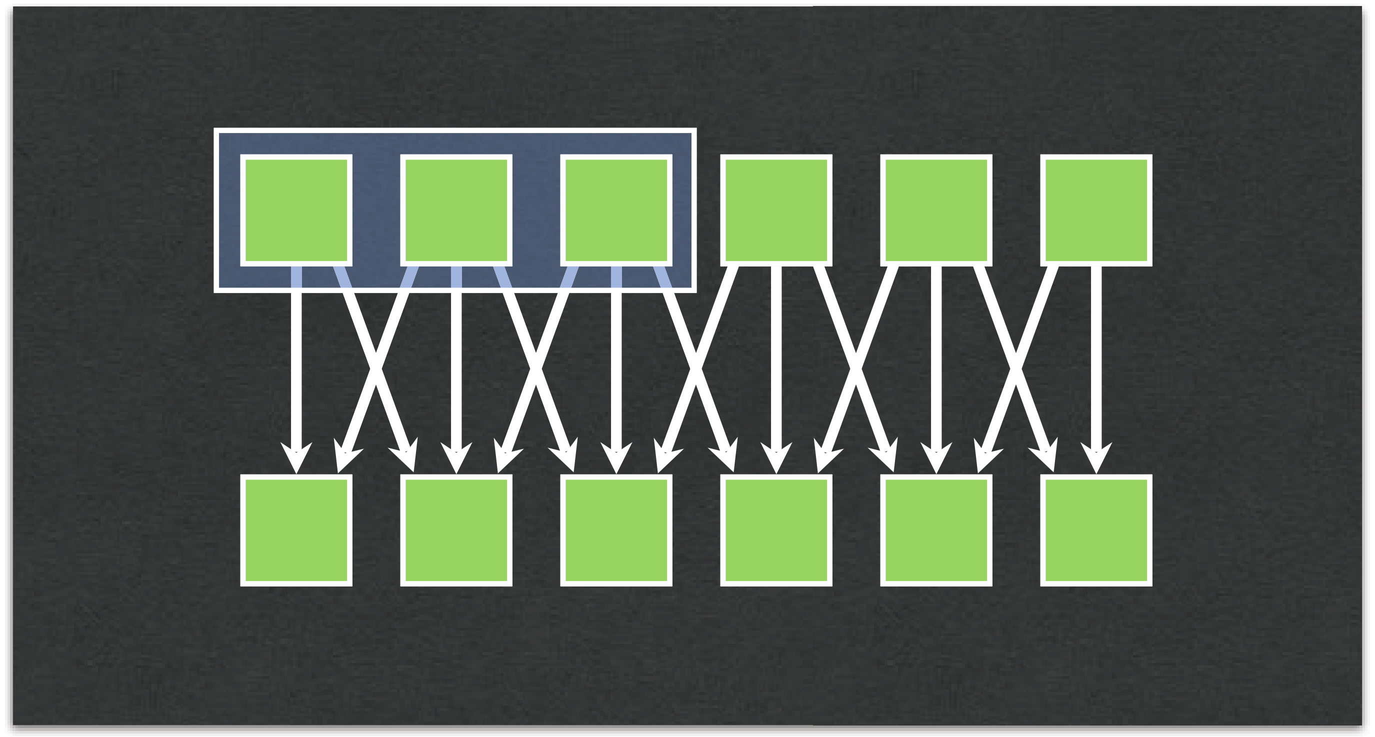
- **Example: Parallelism of on 3 cores**



Too much parallelism



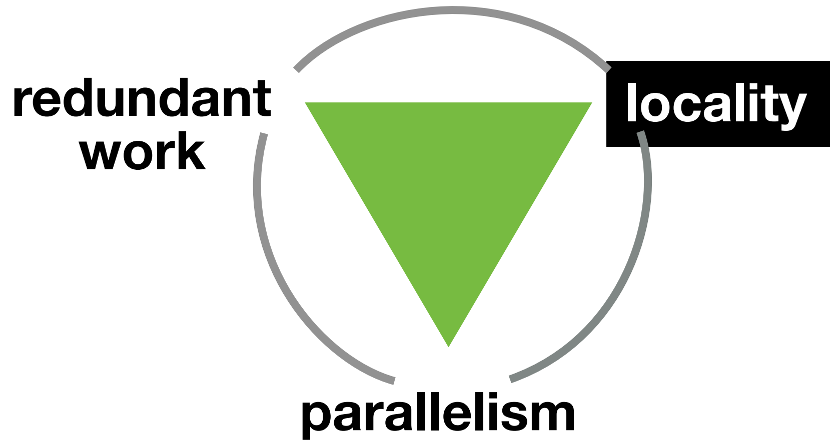
Too little parallelism

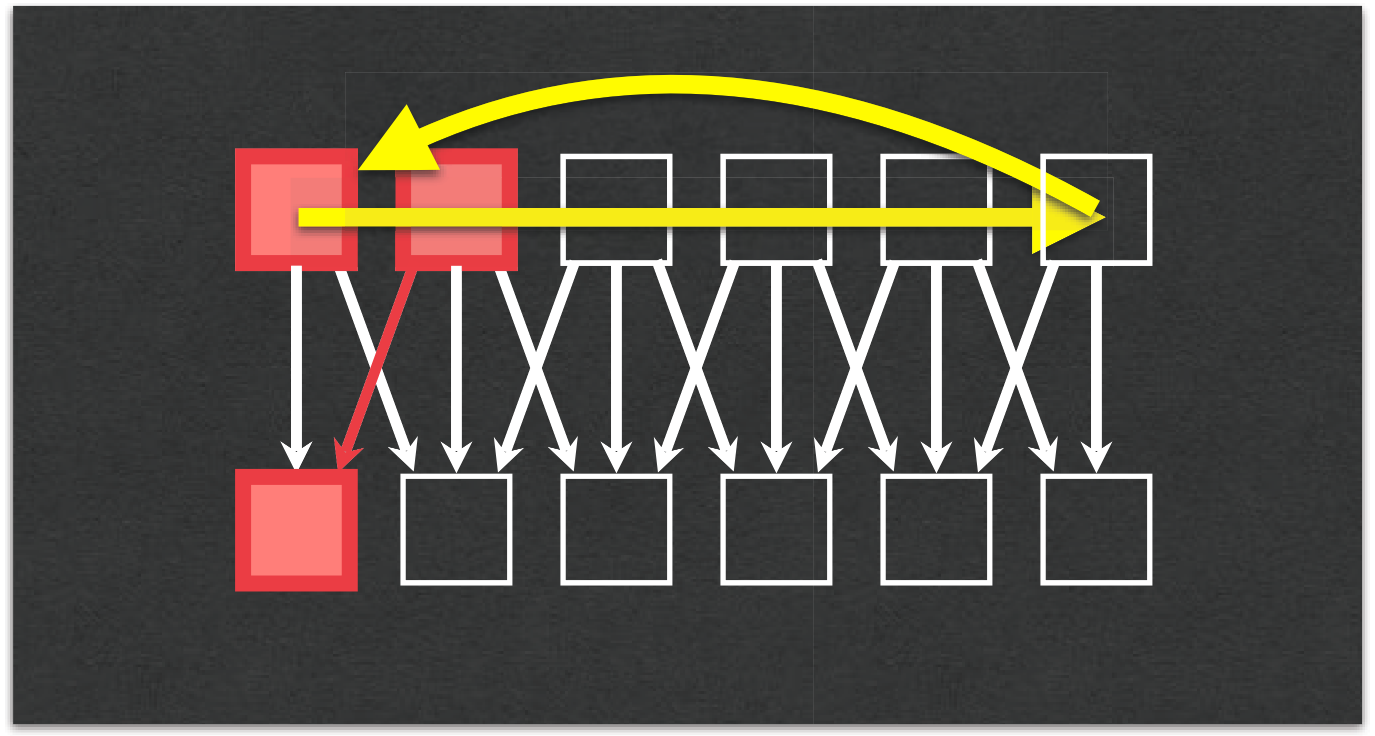


Parallelism just right

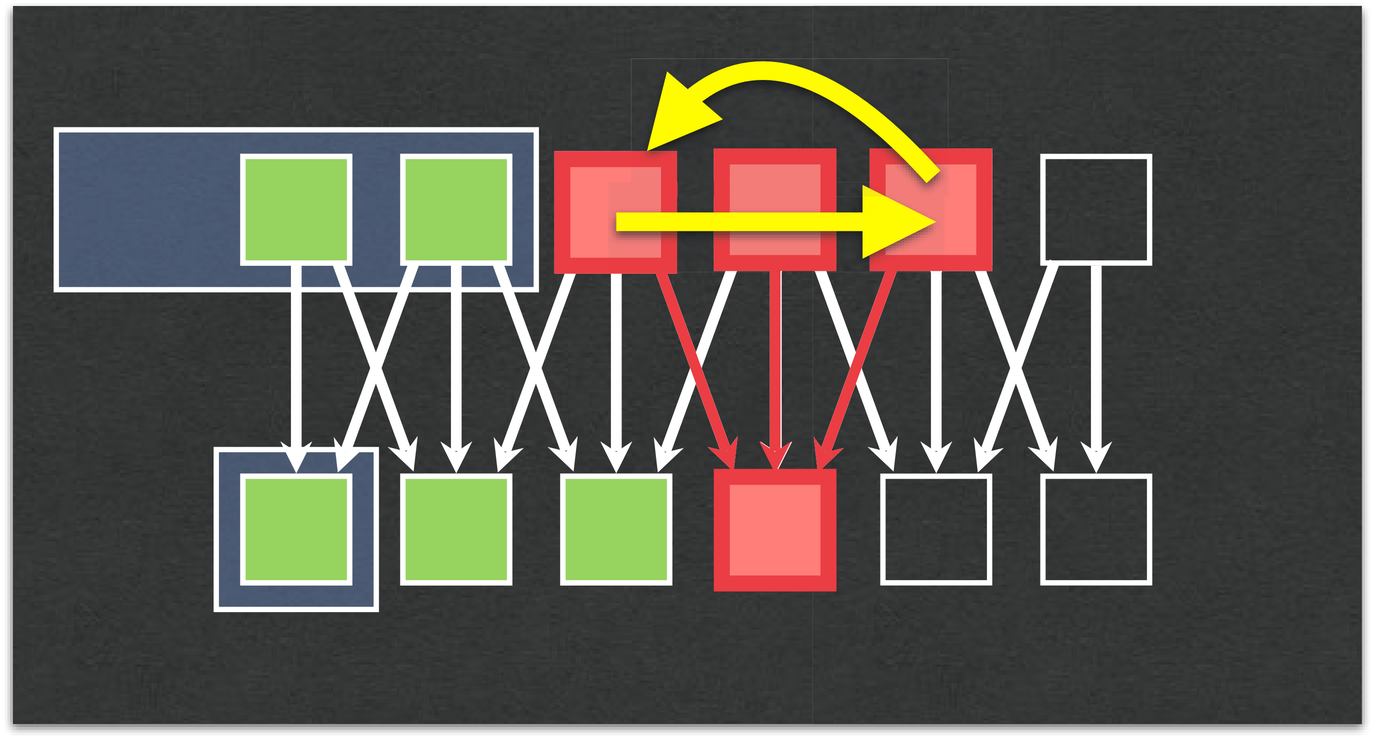
Locality

- **Ones a data is touched, how quickly is it reused**
- **Faster reuse means better cache locality**
- **Locality at multiple levels: registers, L1, L2, LLC**





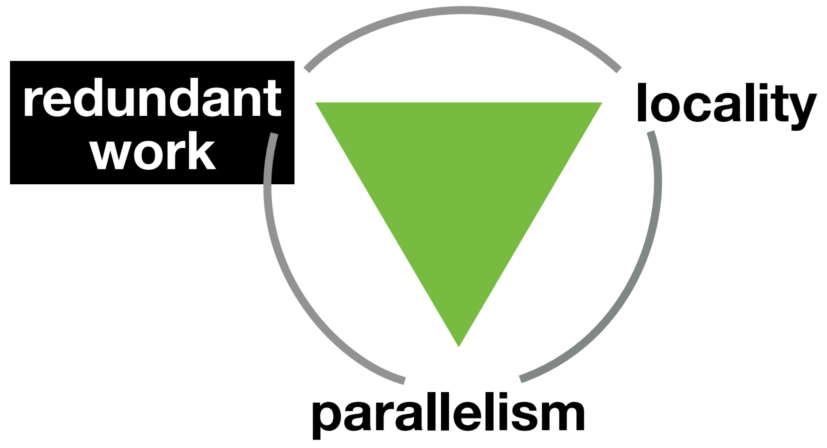
Too little locality

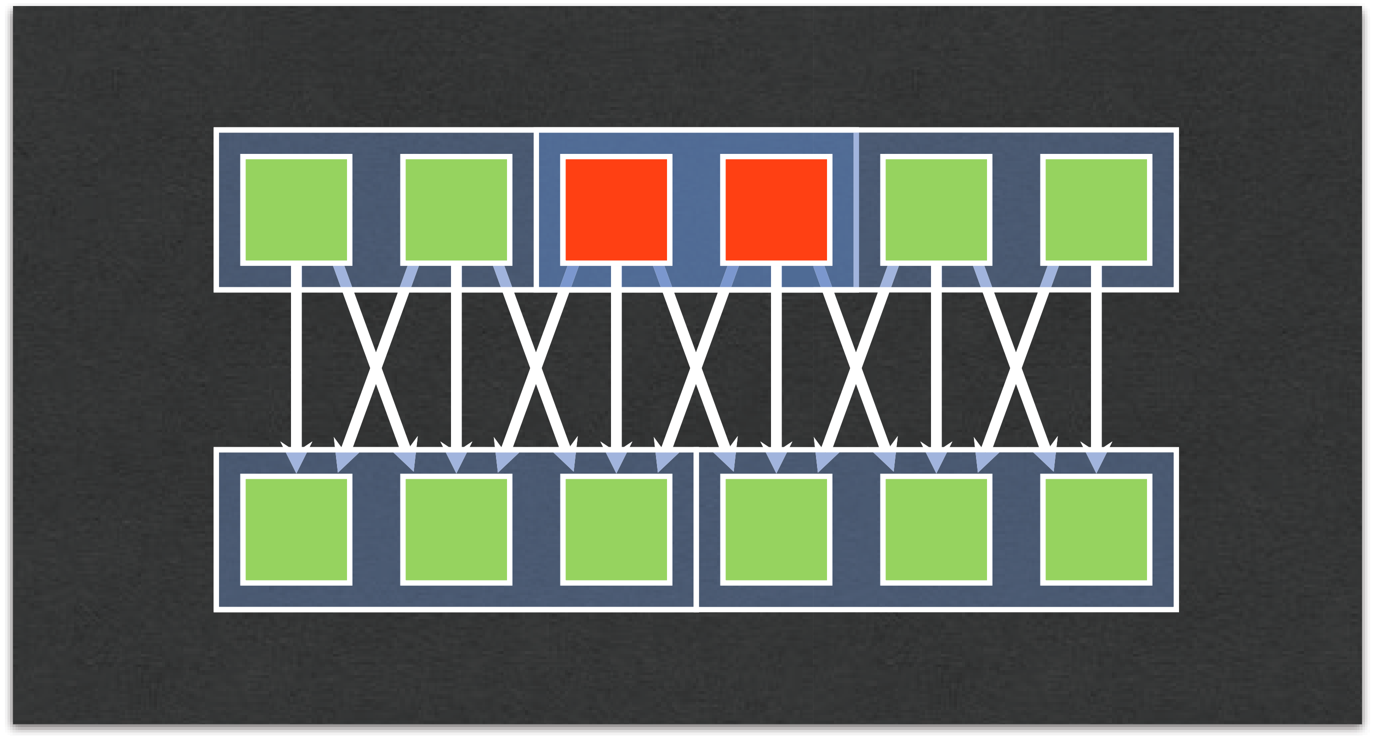


Good Locality

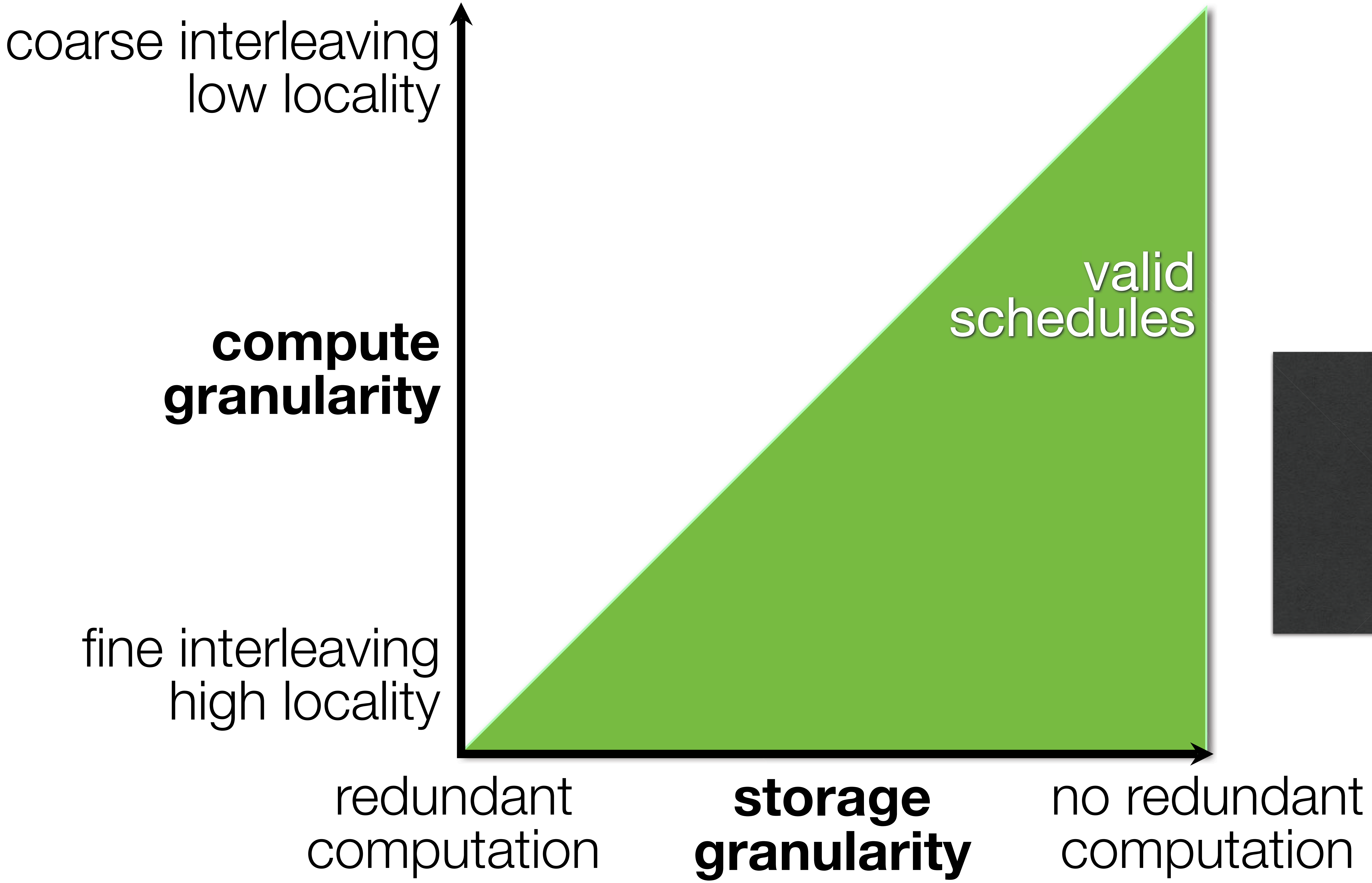
Redundant Work

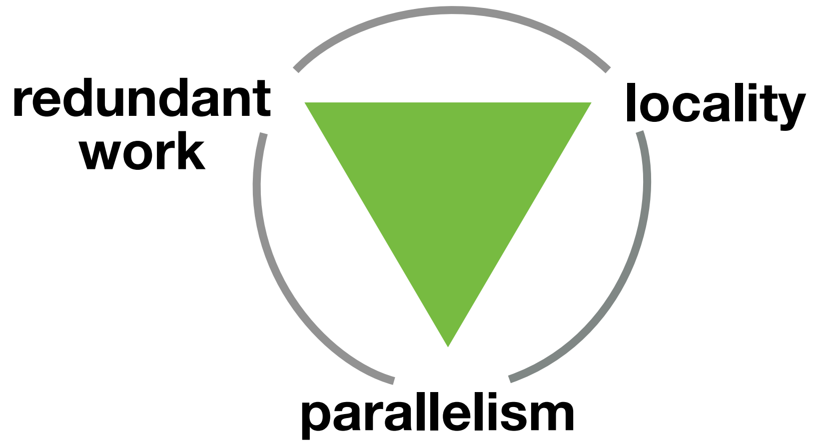
- Sometimes cannot get both locality and parallelism
- A little redundant computation can facilitate both
- Extra cost should be amortizable by the wins



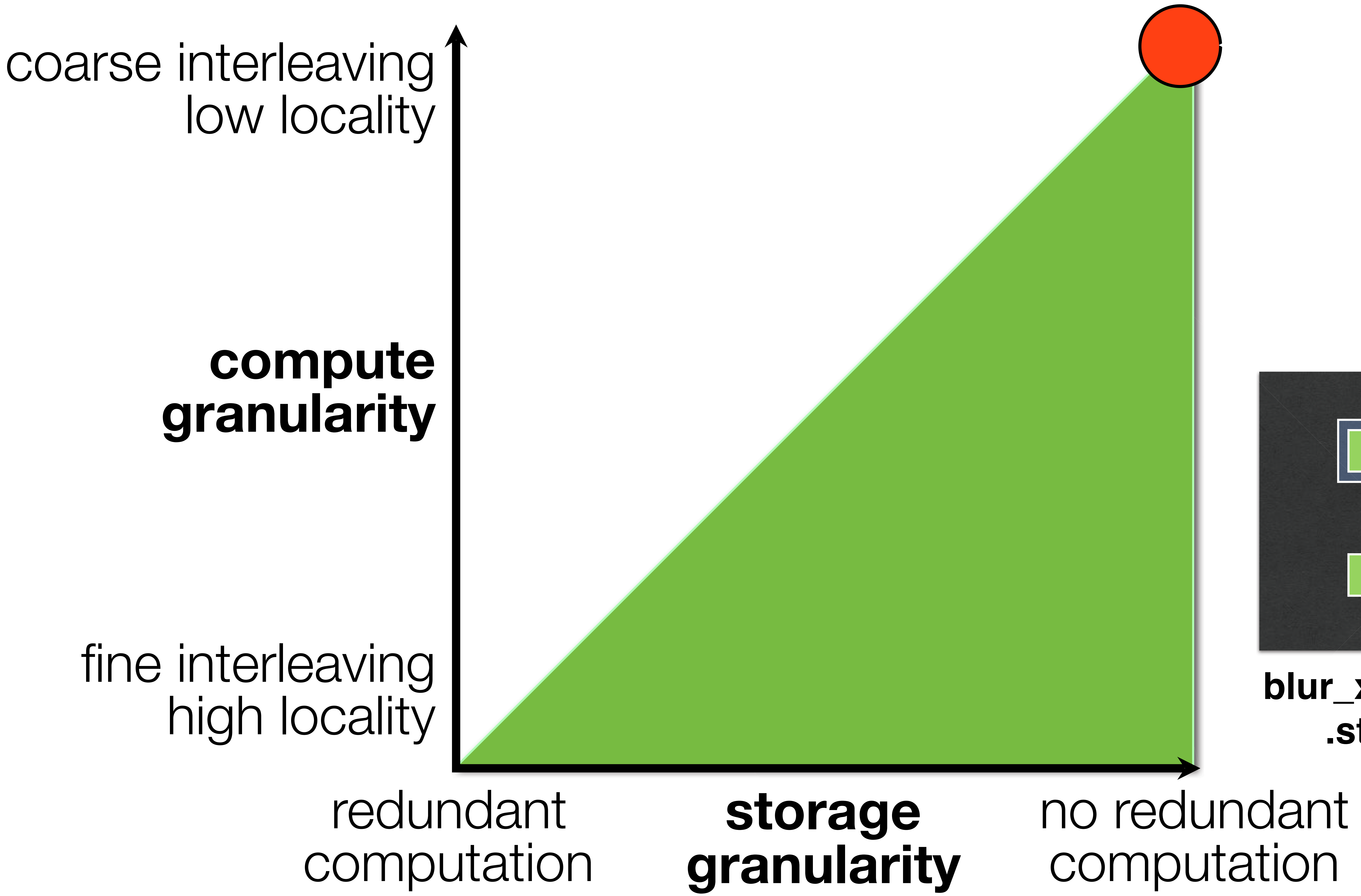


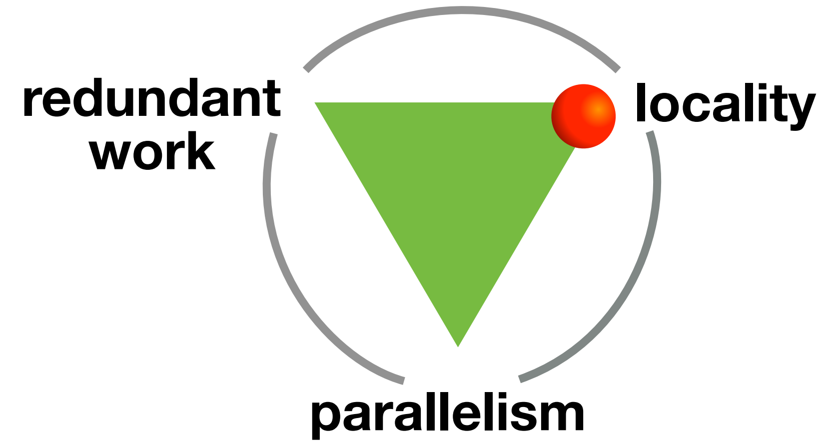
Tradeoff Space Modeled By Granularity Of Interleaving

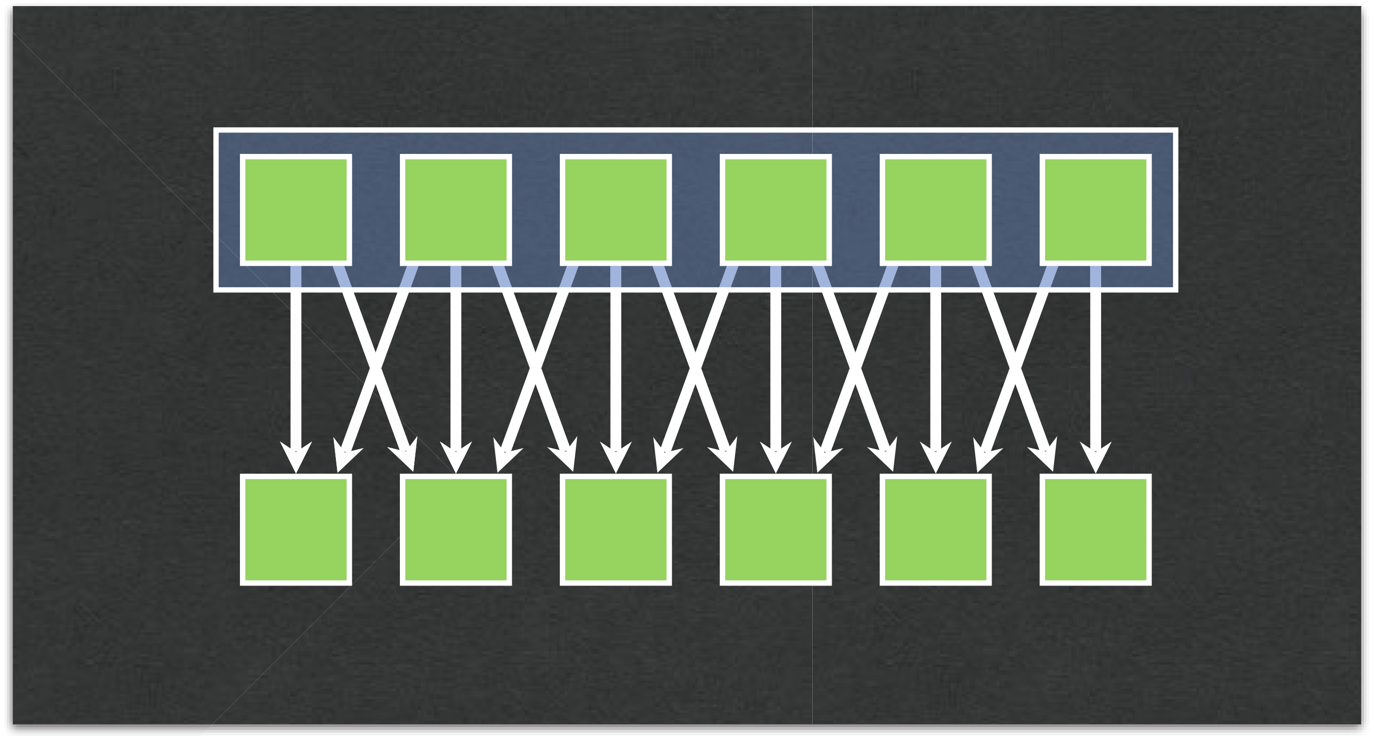




Tradeoff Space Modeled By Granularity Of Interleaving

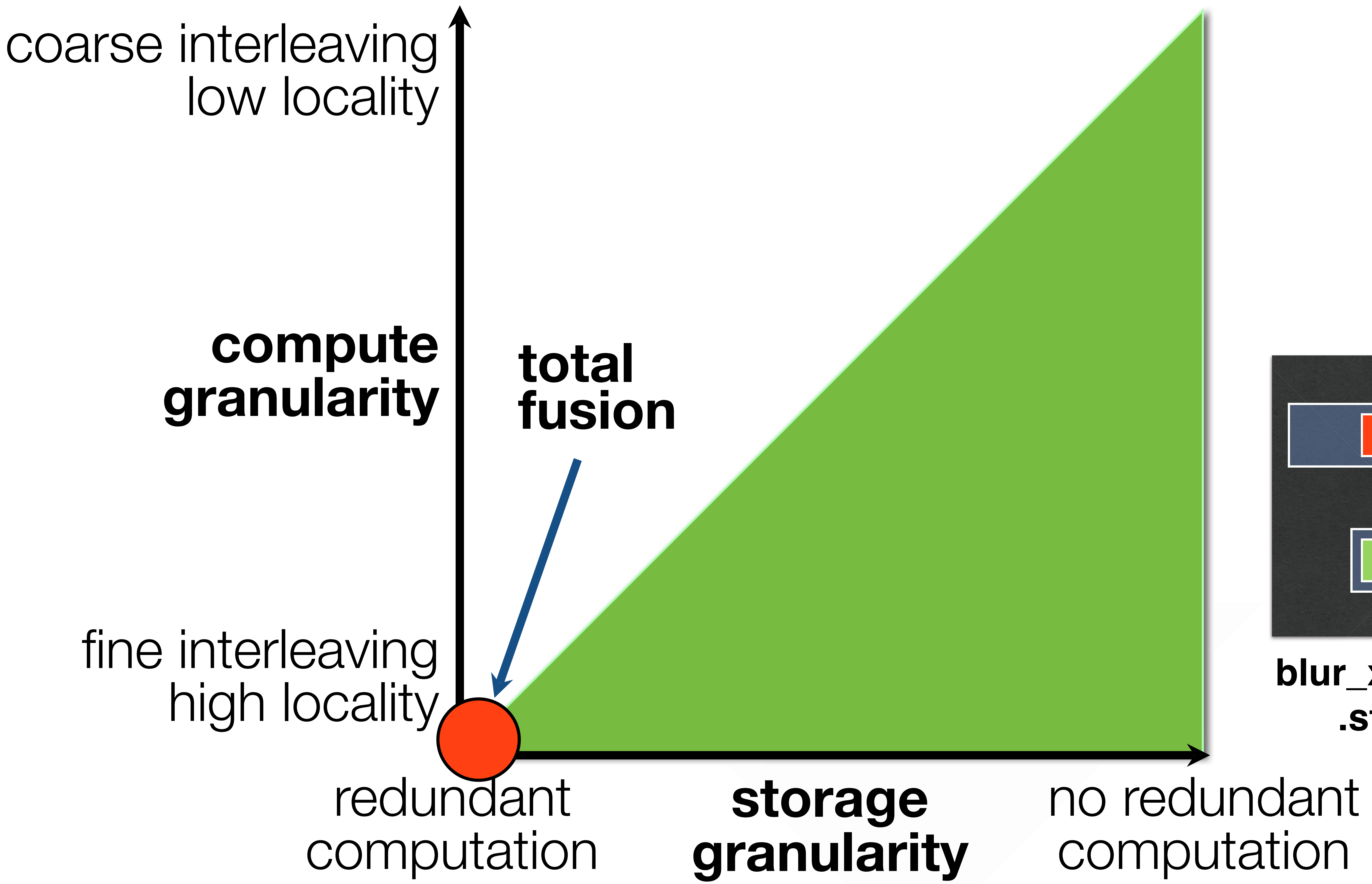


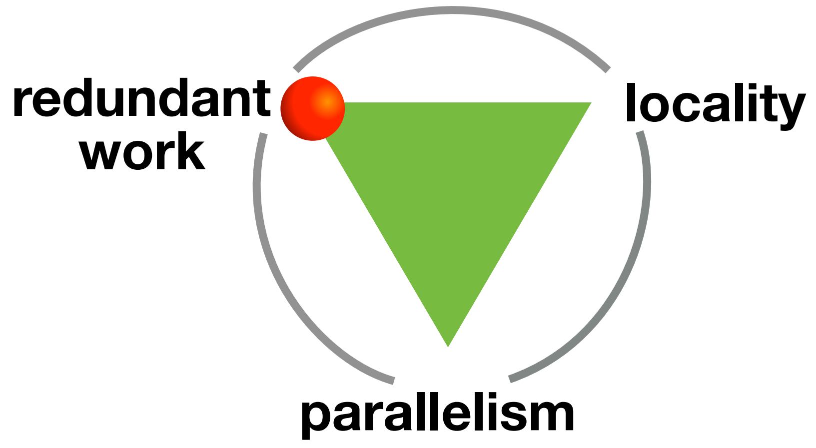


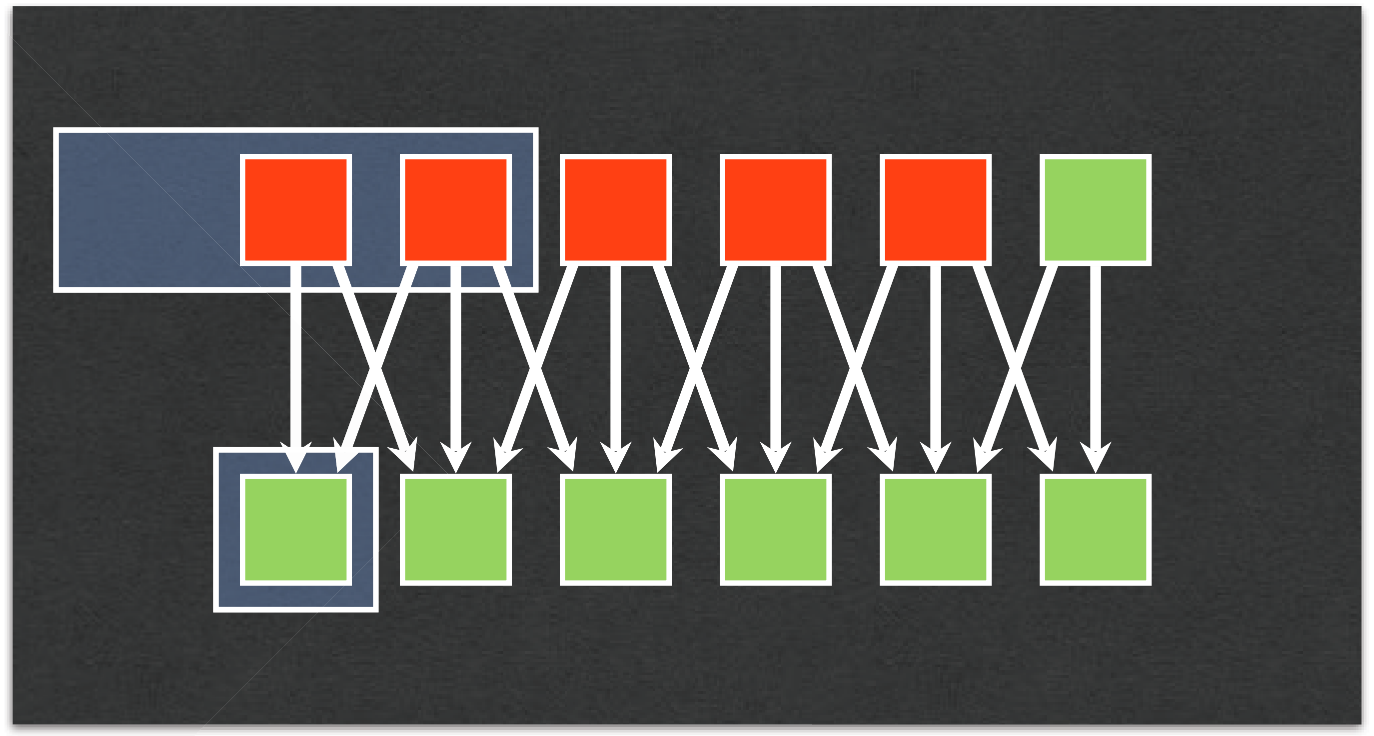


blur_x.compute_at(root)
.store_at(root)

Tradeoff Space Modeled By Granularity Of Interleaving

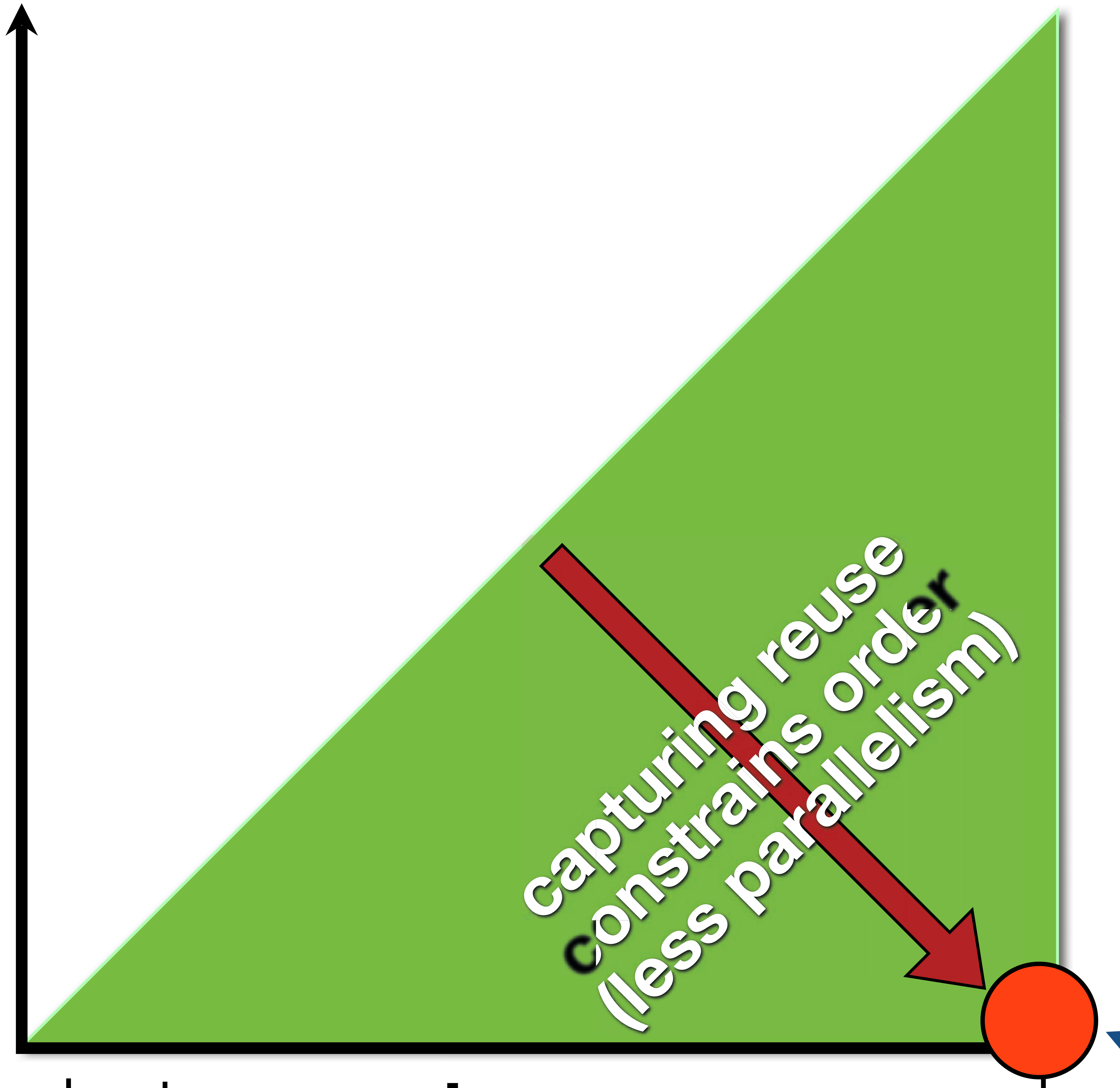


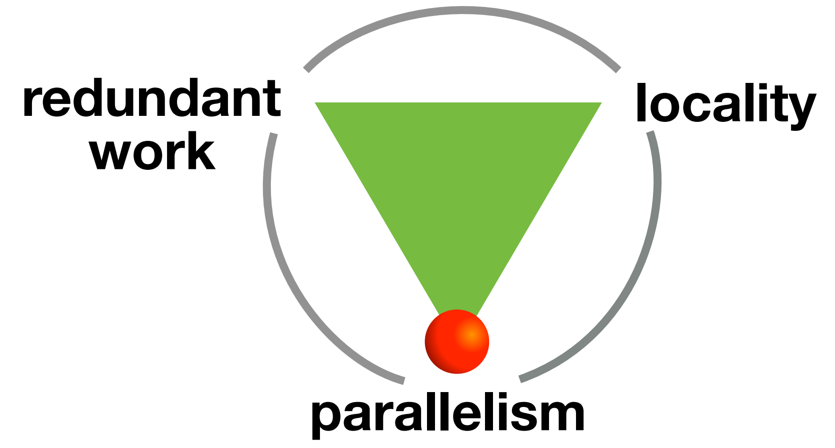


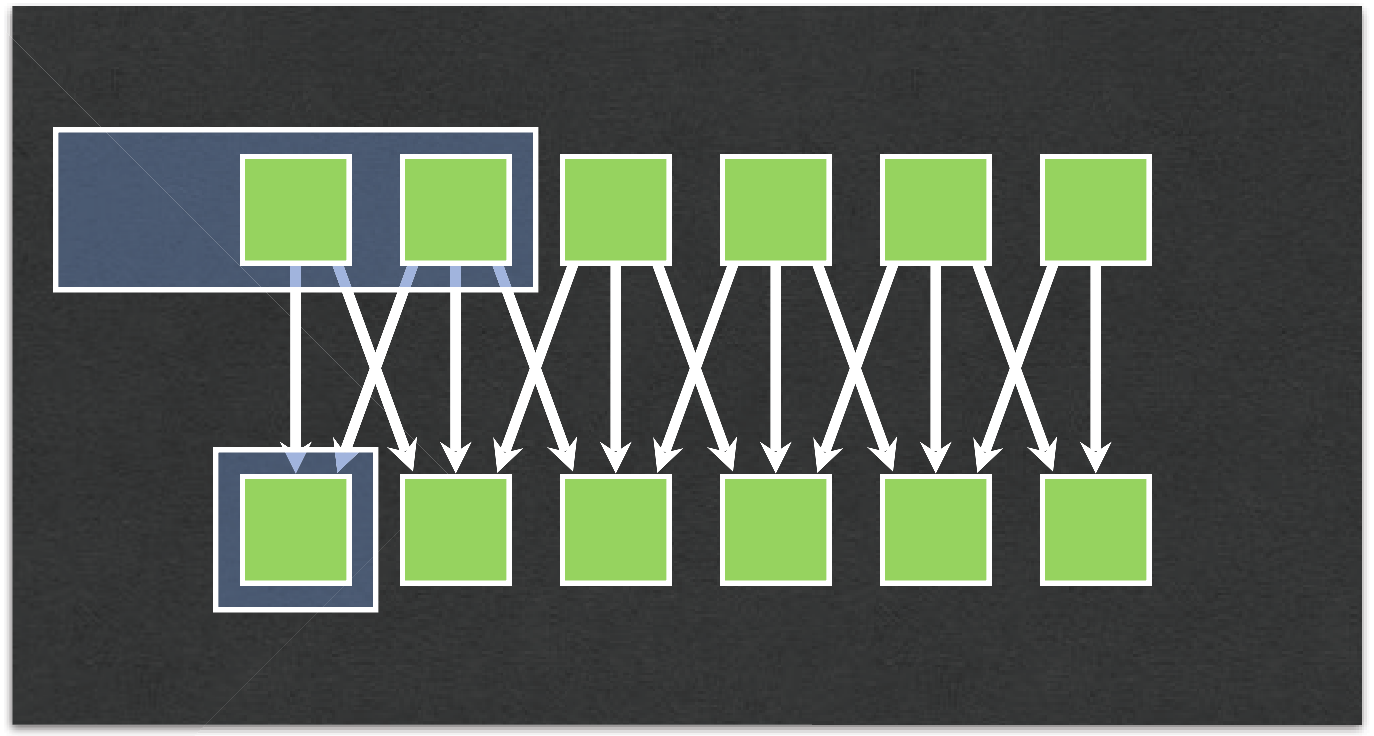


```
blur_x.compute_at(blury, x)
      .store_at(blury, x)
```

Tradeoff Space Modeled By Granularity Of Interleaving

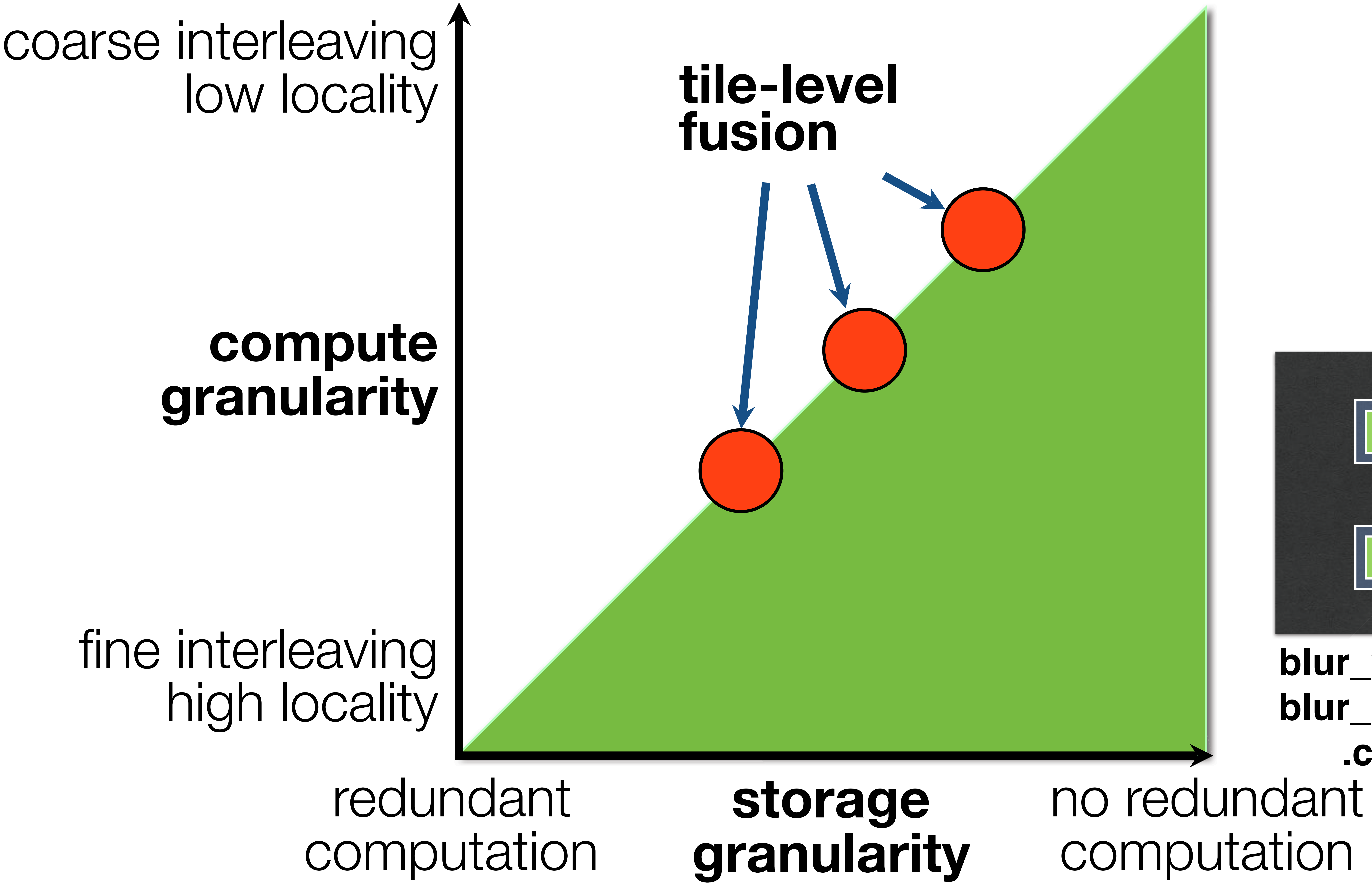


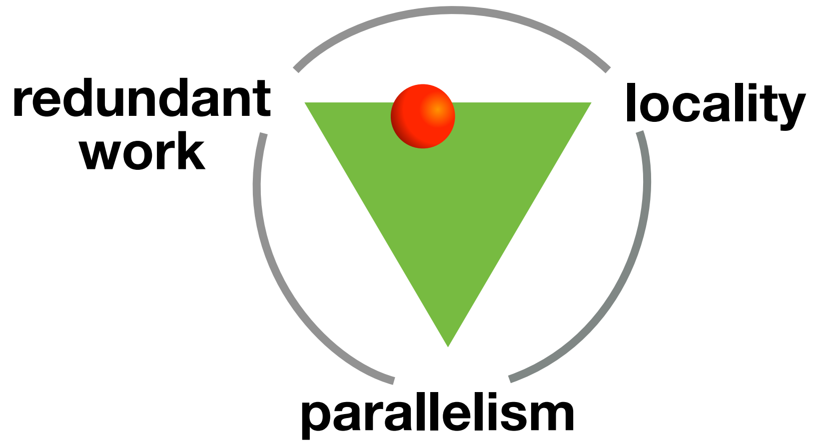


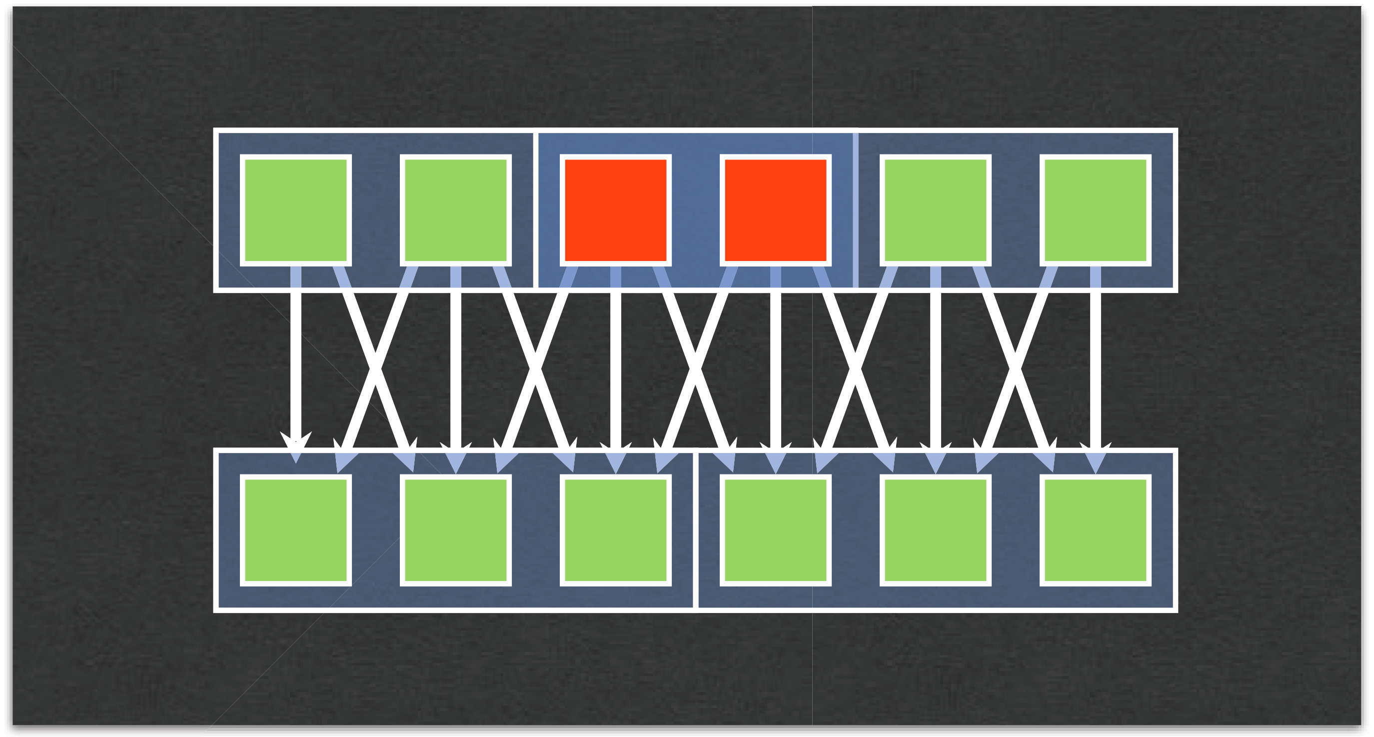


blur_x.compute_at(blury, x)
.store_at(root)

Tradeoff Space Modeled By Granularity Of Interleaving

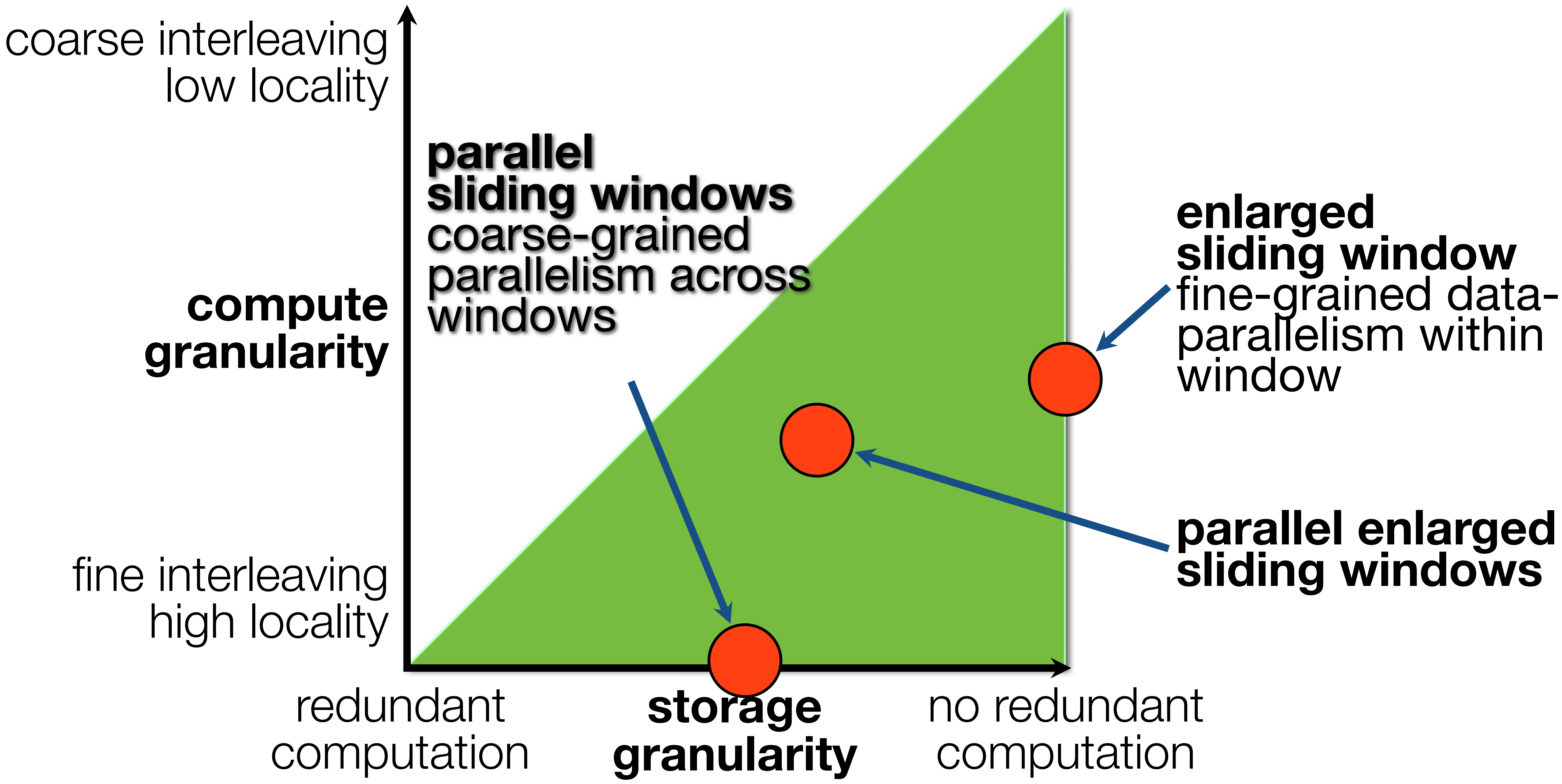






```
blur_y.tile(xo,yo,xi,yi,W,H)
blur_x.compute_at(blury, xo)
      .compute_at(blury, xo)
```

Tradeoff Space Modeled By Granularity Of Interleaving



Schedule Primitives Compose To Create Many Organizations

The Bilateral Grid

[Chen et al. 2007]

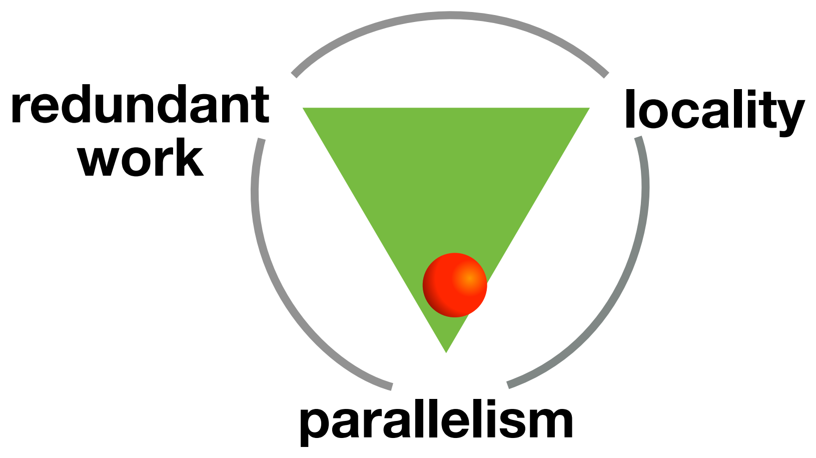
An accelerated bilateral filter

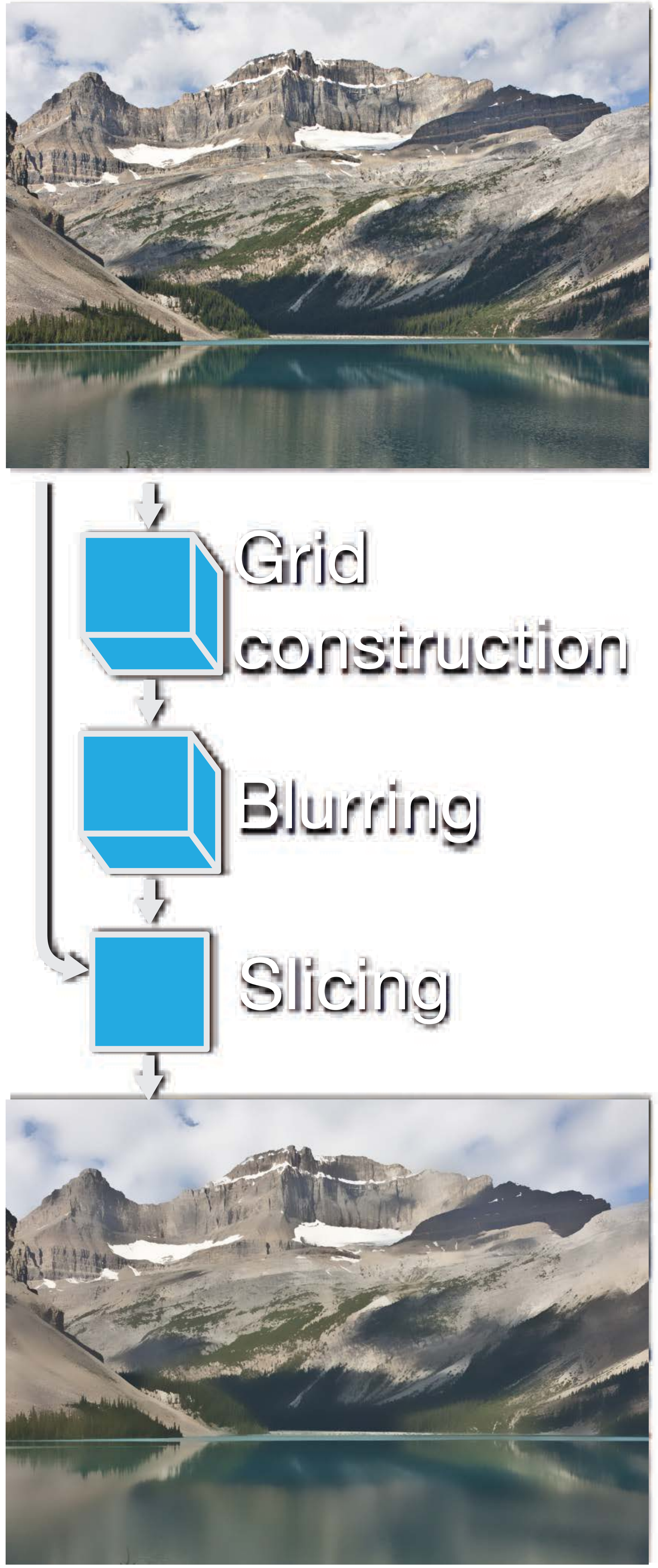
Original: 122 lines of (clean) C++

Halide: 34 lines of algorithm

On the CPU, 5.9x faster

On the GPU, 2x faster than Chen's hand-written CUDA version



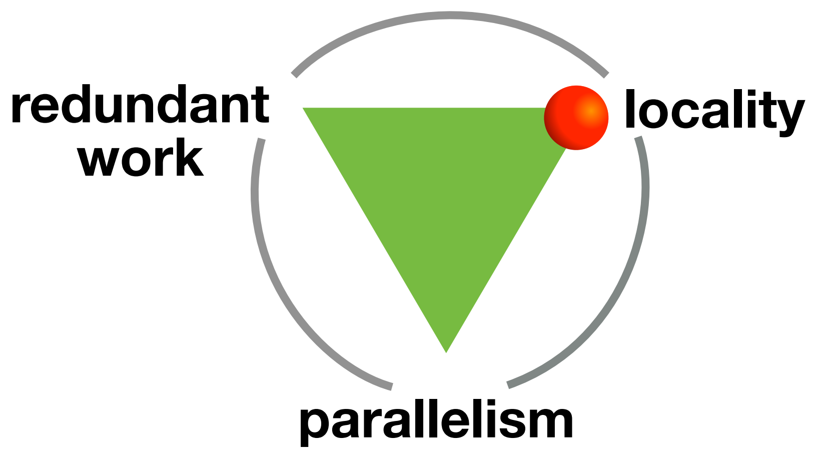


“Snake” Image Segmentation

[Li et al. 2010]

Segments objects in an image using level-sets

Original: 67 lines of MATLAB

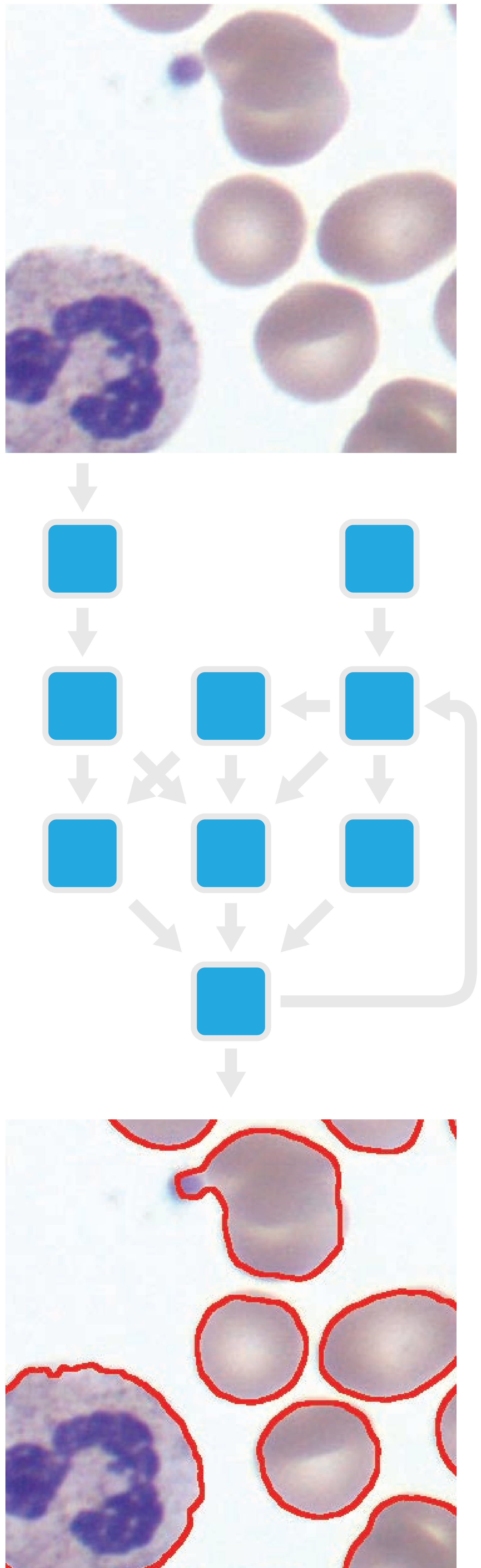


Halide: 148 lines of algorithm

On the CPU, 70x faster

MATLAB is memory-bandwidth limited

On the GPU, 1250x faster



Local Laplacian Filters

prototype for Adobe Photoshop Camera Raw / Lightroom

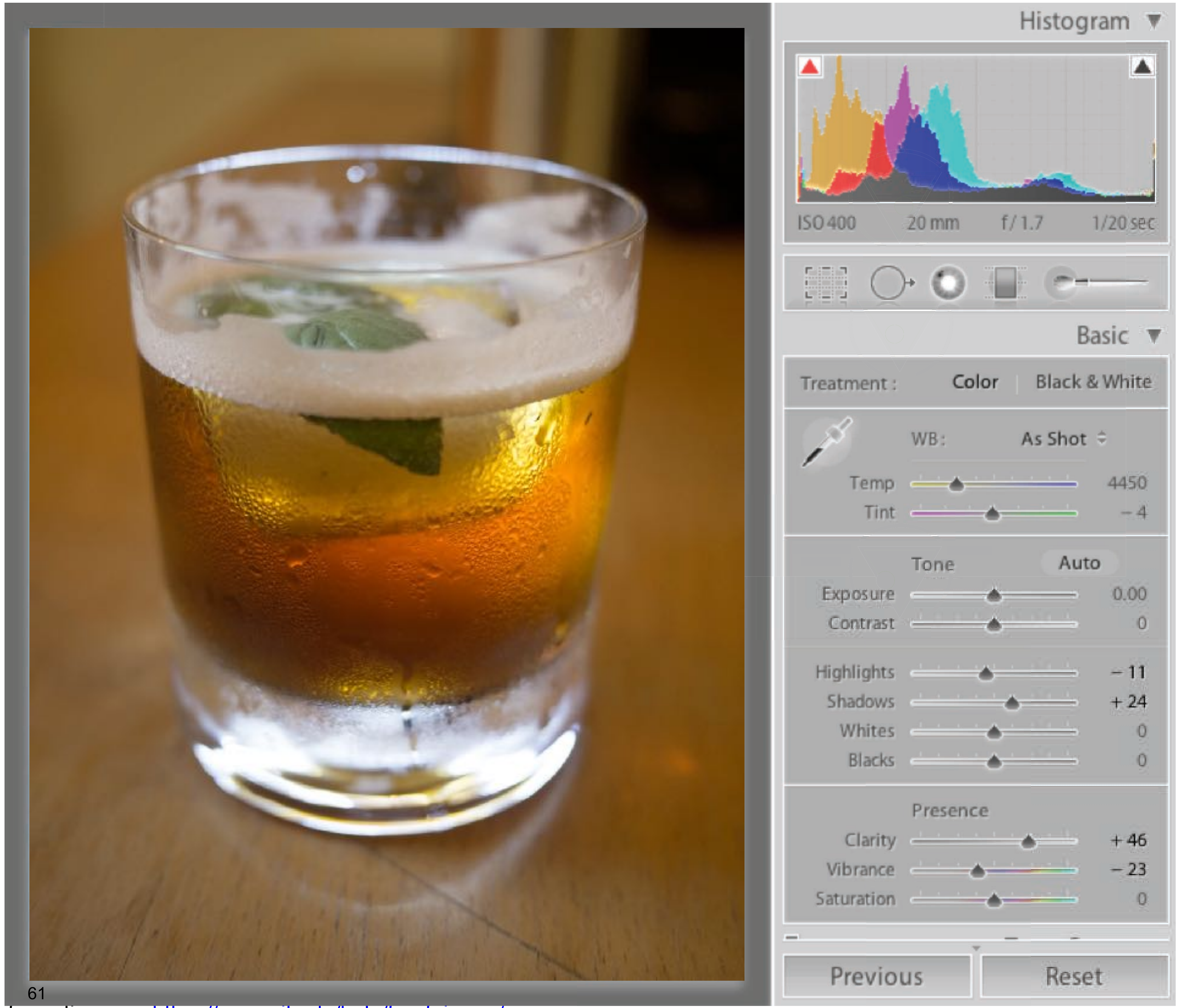
Reference: 300 lines C++

Adobe: 1500 lines
3 months of work
10x faster (vs. reference)

Halide: 60 lines
1 intern-day

20x faster (vs. reference)
2x faster (vs. Adobe)

GPU: 90x faster (vs. reference)
9x faster (vs. Adobe)



Real-World Adoption

Data center
Android
Browser
Glass
G Photos auto-enhance
HDR+
> 200 pipelines, 10s of kLOC in production

Photoshop in IOS

Front end language for the
Snapdragon Image Signal Processor

Halide touches every video uploaded.
65B frames/day

OpenTuner

- Performance Engineering is all about finding the right:
  - block size in matrix multiply (voodoo parameters)
  - strategy in the dynamic memory allocation project
  - flags in calling GCC to optimize the program
  - schedule in Halide
  - schedule in GraphIt

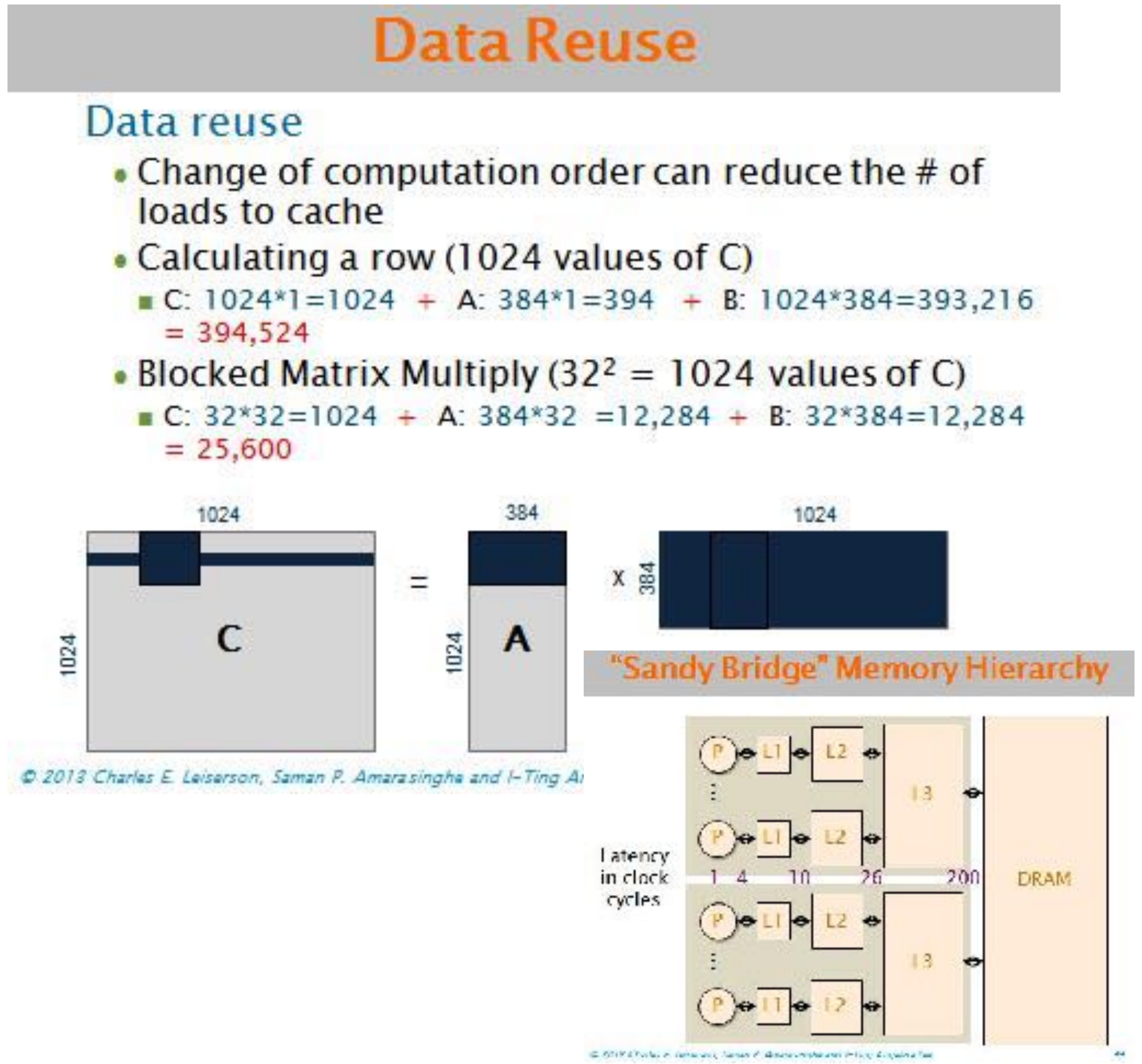
How to find the right value

1. Model-Based
2. Heuristic-Based
3. Exhaustive Search
4. Autotuned (OpenTuner)

1. Model Based Solutions

Come-up with a comprehensive model

- In this case, a model for the memory system and data reuse



Pros:

- Can explain exactly why we chose a given tile size
- "Optimal"

Cons:

- Hard to build models
- Cannot model everything
- Our model may miss an important component

2. Heuristic Based Solutions

“A rule of thumb” that works most of the time

- In this case, small two-to-the-power tile sizes works most of the time
- Hard-code them (eg: S = 8)

Pros

- Simple and easy to do
- Works most of the time

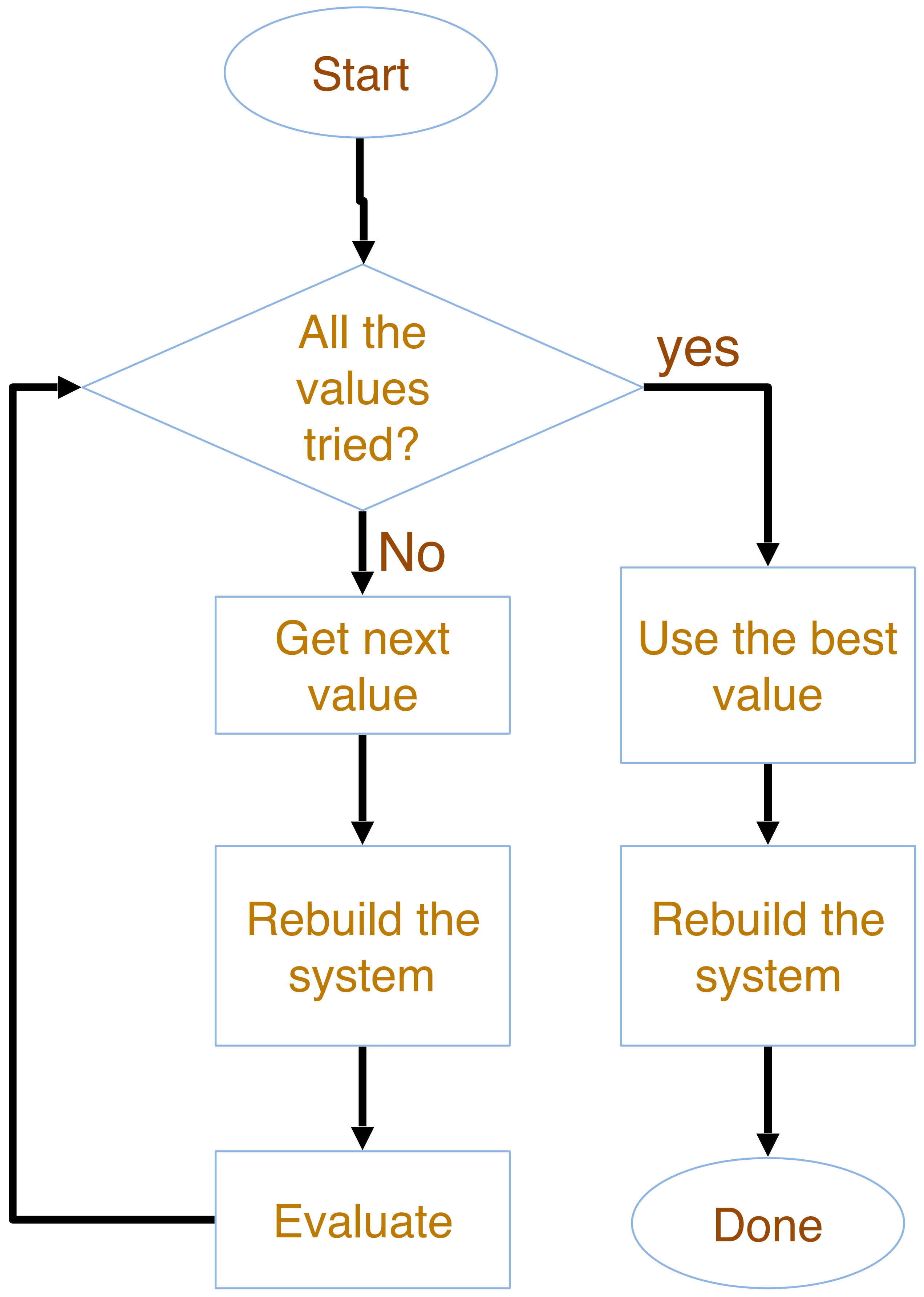
Cons

- Simplistic
- However, always suboptimal performance
- In some cases may be really bad

3. Exhaustive Search

Empirically evaluate all the possible values

- All possible integers for S



Pros:

- Will find the “optimal” value

Cons:

- Only for the inputs evaluated
- Can take a looooong time!
  - Prune the search space
    - Only integers that are powers-of-2 from vector register size to the cache size?

4. Autotuning based solutions

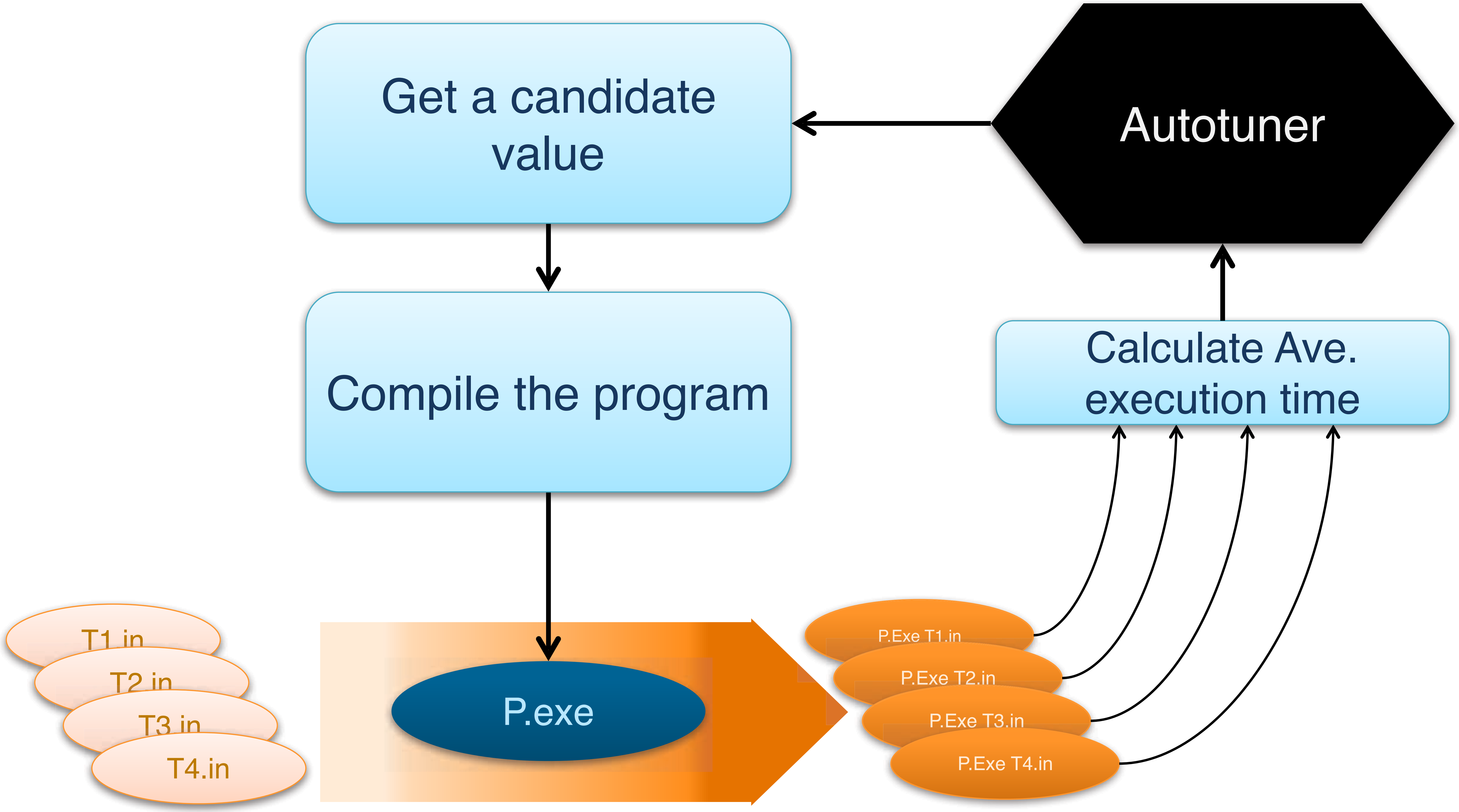
What?

1. Define a space of acceptable values
2. Choose a value at random from that space
3. Evaluate the performance given that value
4. If satisfie … ormanc
   time limit exceeded → finish
5. Choose a new value from the feedback
6. Goto 3

When?

How?

Autotuning A Program



Ensembles of techniques

Nelder-Mead Simplex (Hill climber)

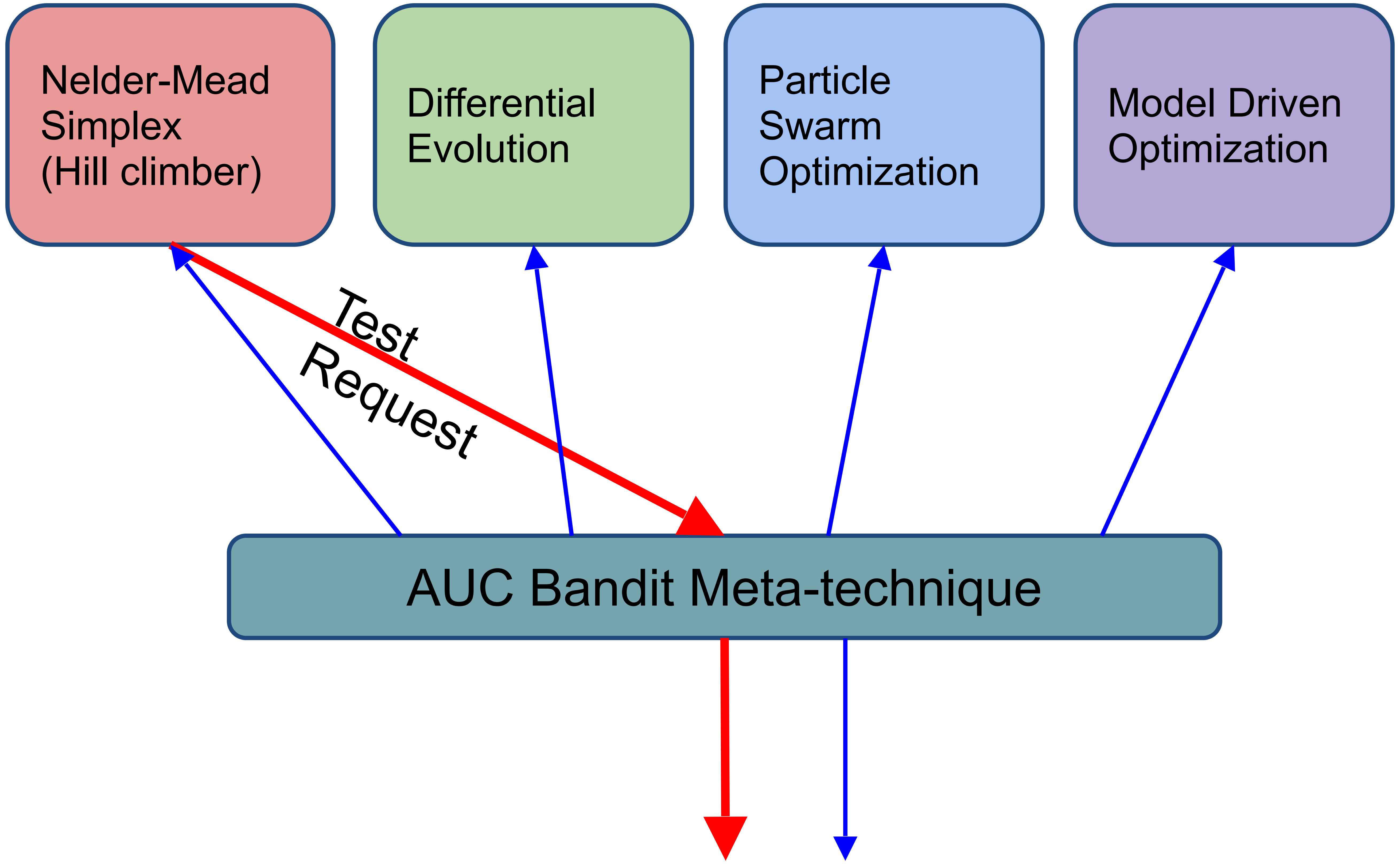
Differential Evolution

Particle Swarm Optimization

Model Driven Optimization

- Many different techniques
- Each best suited to solve different problems
- Hard to write a single autotuner that performs well in different domains
- Can we make these techniques work together?

Ensembles of techniques



- Meta-technique divides testing budget between sub-techniques
- Results are shared between all techniques

Autotuning GraphIt

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:11
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```

Finding the best schedule can be hard for non-experts.

Scheduling Functions

```
schedule:
   program->configApplyDirection("s1", "DensePull");
   program->configApplyParallelization("s1", "dynamic-vertex-parallel");
   program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Goal

Algorithm Specification

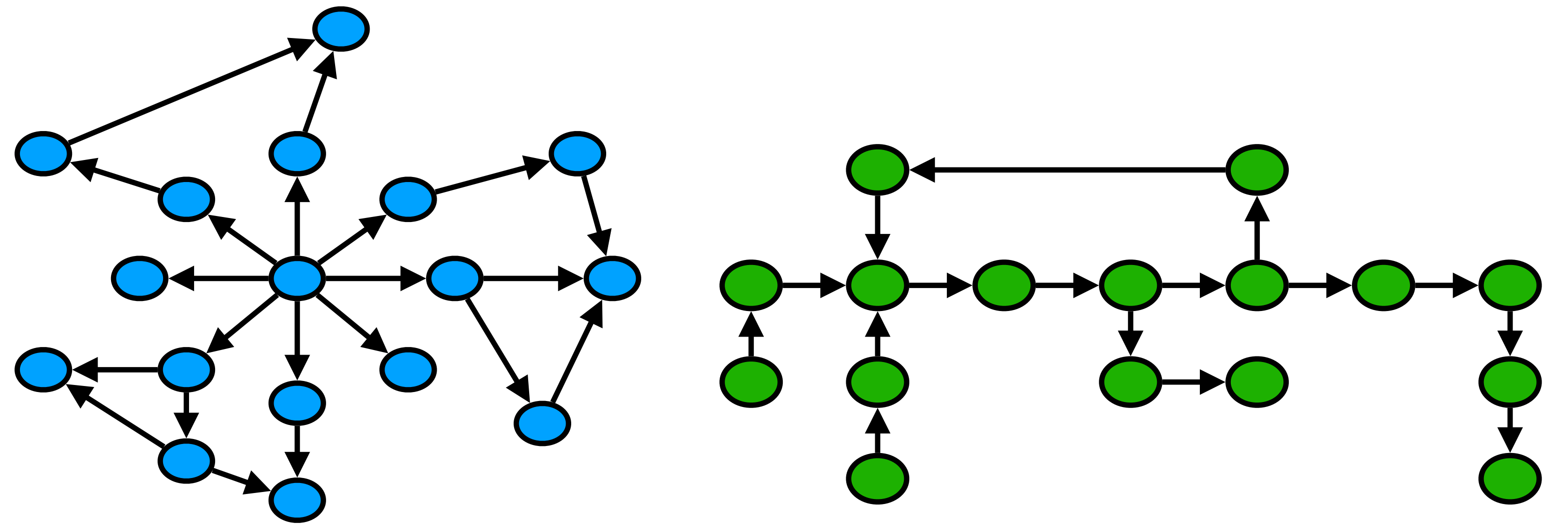
```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end

func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

 func main()
    for i in 1:11
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
    end
 end
```

Ideally, the user only need to write the algorithm

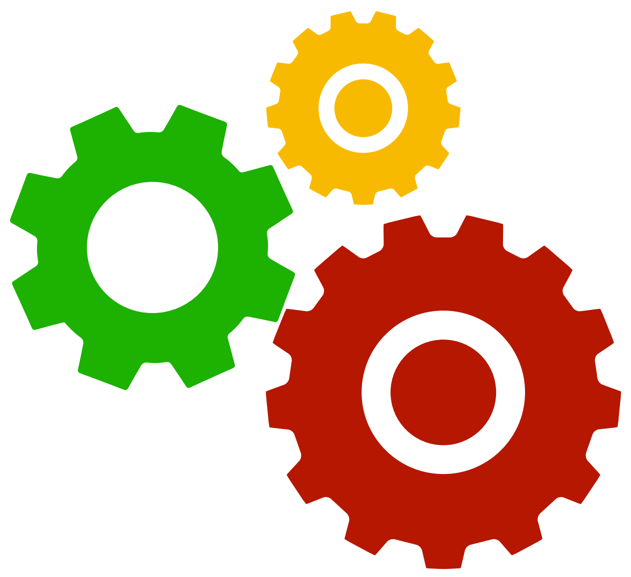
Autotuner

Algorithm Specification

```
func updateEdge (src: Vertex, dst: Vertex)
   new_rank[dst] += old_rank[src] / out_degree[src]
end


func updateVertex (v: Vertex)
   new_rank[v] = beta_score + 0.85*new_rank[v];
   old_rank[v] = new_rank[v];
   new_rank[v] = 0;
end

func main()
   for i in 1:11
      #s1# edges.apply(updateEdge);
      vertices.apply(updateVertex);
   end
end
```



Input Graphs

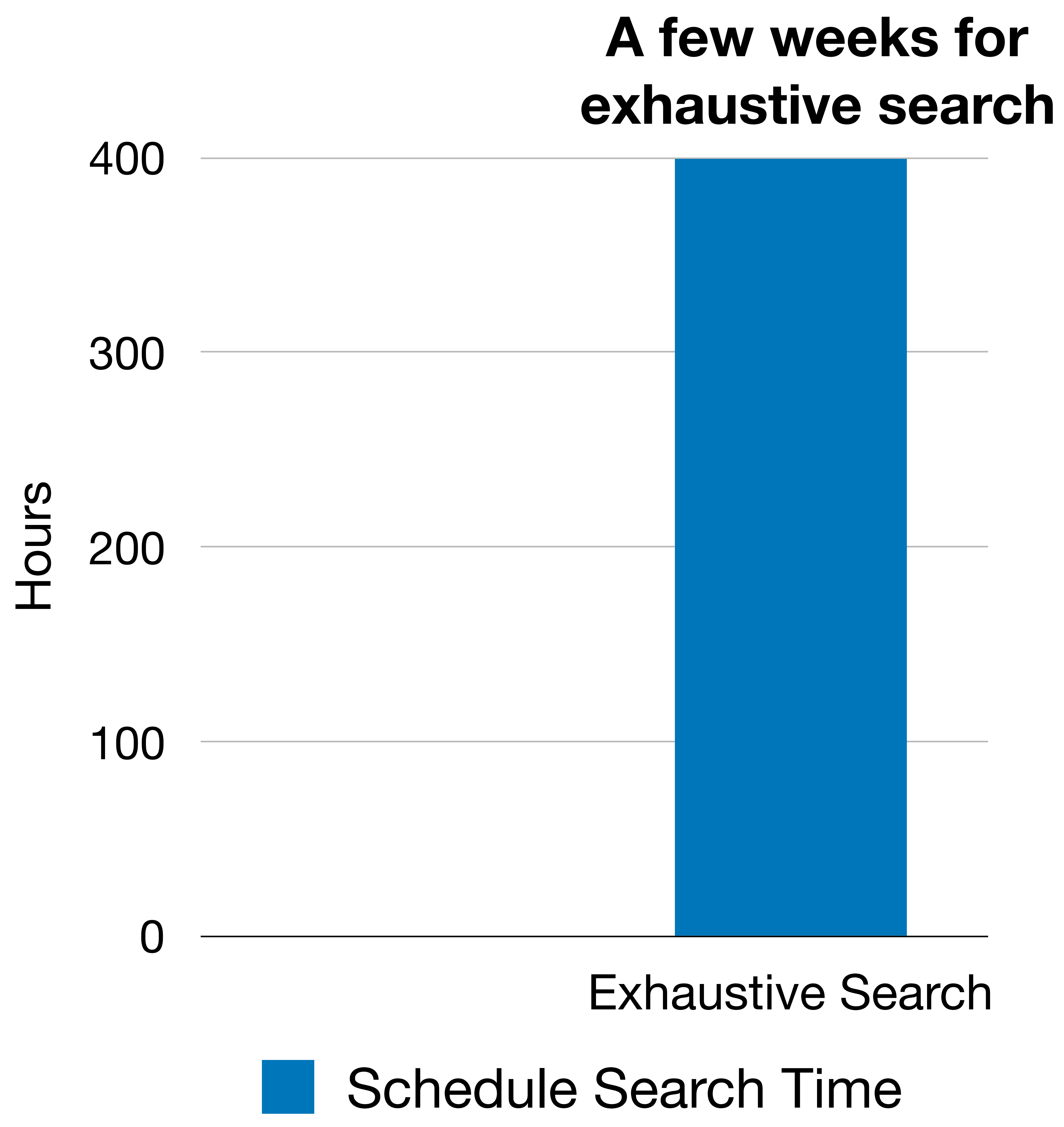


Autotuner

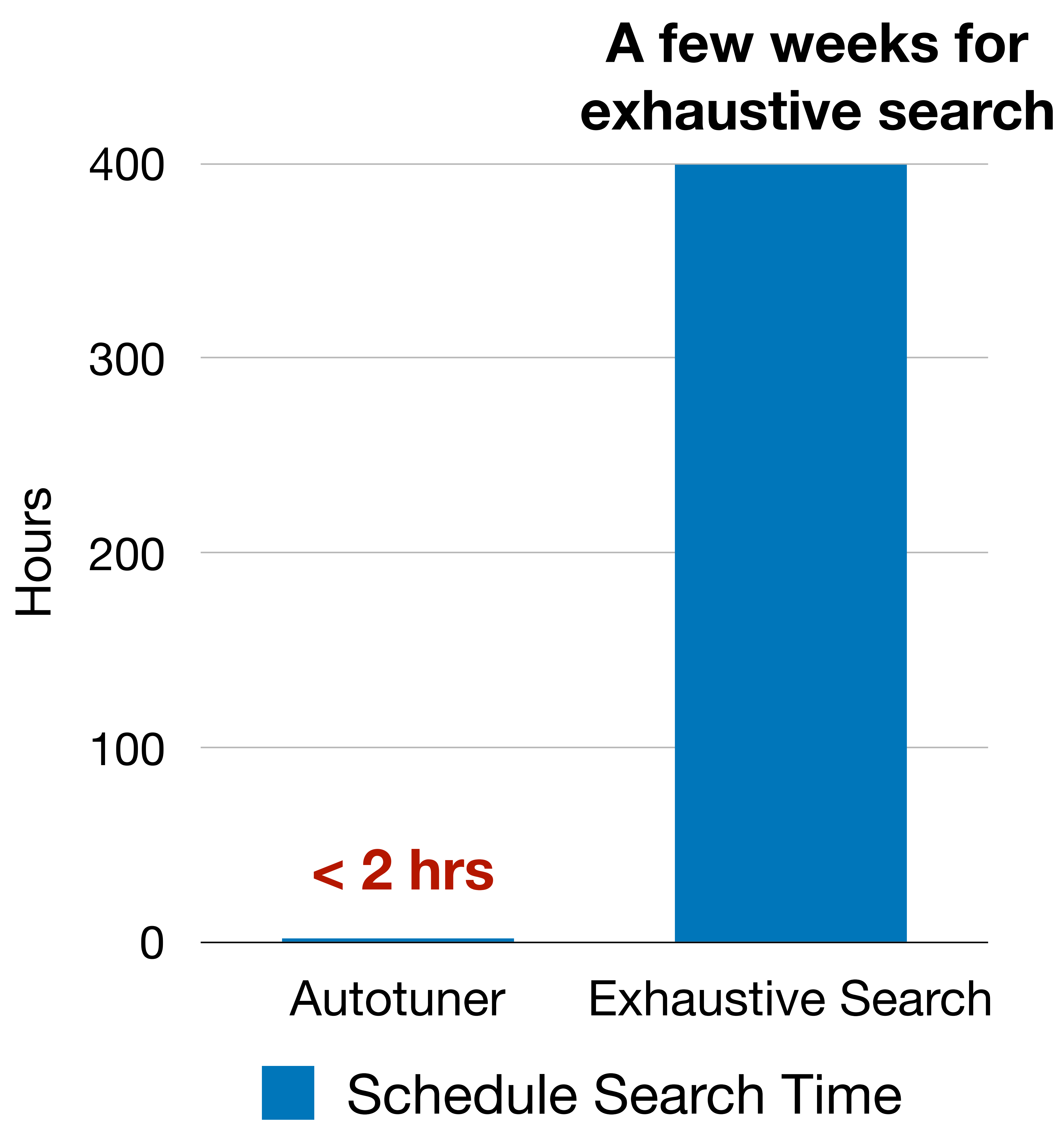
Scheduling Functions

```
schedule:
   program->configApplyDirection("s1", "DensePull");
   program->configApplyParallelization("s1", "dynamic-vertex-parallel");
   program->configApplyNumSSG("s1", "fixed-vertex-count", 10);
```

Autotuner

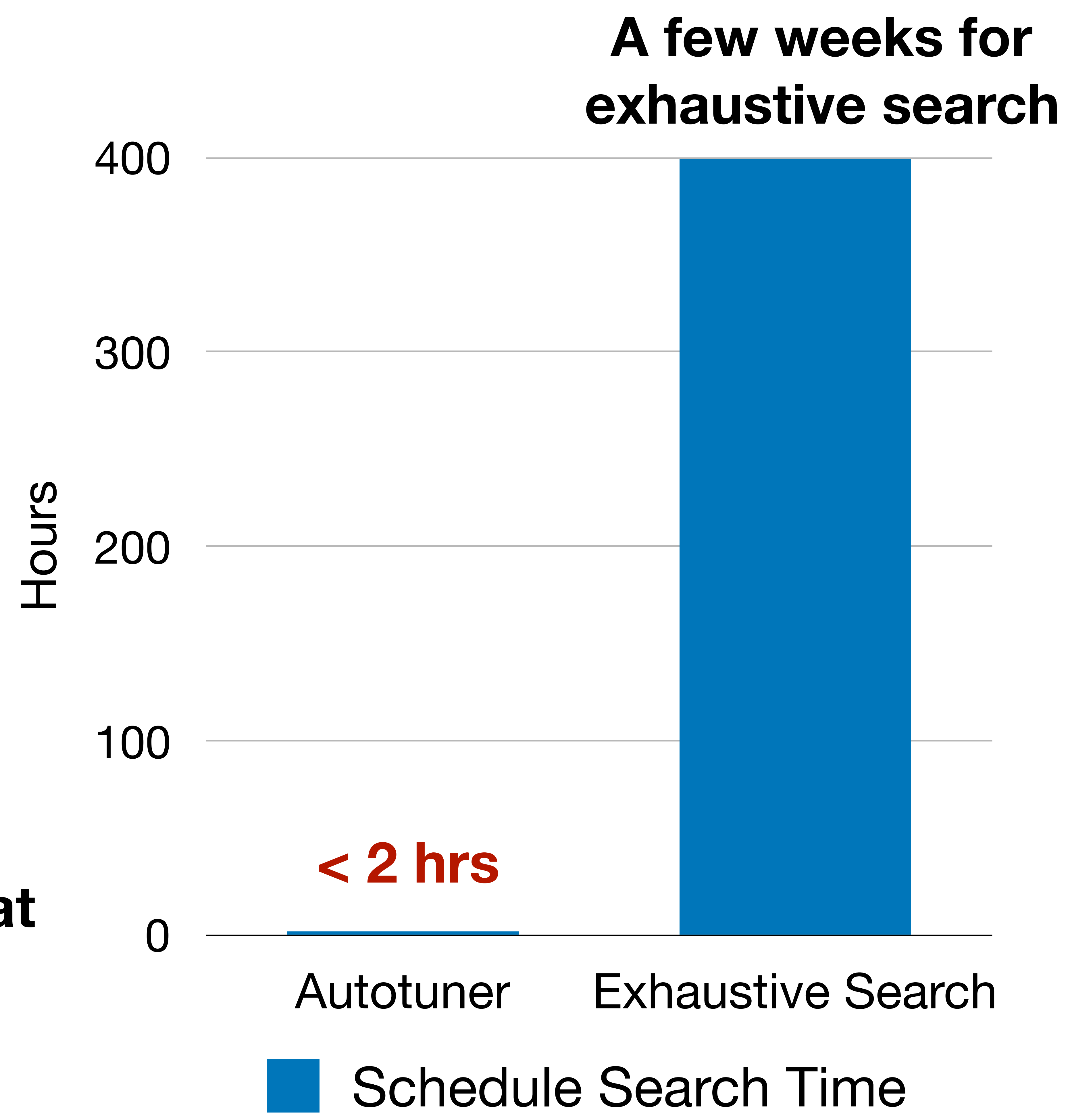

A few weeks for exhaustive search
400
300
200
100
0
Hours
Exhaustive Search
Schedule Search Time

Autotuner



Uses an ensemble of search methods. Build on top of OpenTuner [PACT14]

Autotuner



Finds a few schedules that outperform hand-tuned schedules