AMATH 483/583
High Performance Scientific Computing

Lecture 13:
Case Studies: TwoNorm, PageRank, Lambda

Xu Tony Liu, PhD
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, WA

Overview

- In our last episode
    - Race condition (the critical-section problem)
    - Solutions of race condition
    - Mutex, Deadlock, Lock_guard, std::lock (avoid deadlock)
    - Asynchronous operation (std::async and std::future)
    - std::atomic (only working with integral type)
- Two Norm
- Lambda anonymous functions
- PageRank

Race Condition

```cpp
double pi = 0.0;

void pi_helper(int begin, int end, double h) {
  for (int i = begin; i < end; ++i)
    pi += (h*4.0) / (1.0 + (i*h*i*h));
}

int main(int argc, char* argv[]) {
  int N = 1024 * 1024; double h = 1.0/ (double)N;

  std::thread t0(pi_helper, 0,      N/4,   h);
  std::thread t1(pi_helper, N/4,    N/2,   h);
  std::thread t2(pi_helper, N/2,    3*N/4, h);
  std::thread t3(pi_helper, 3*N/4, N,      h);

  t0.join();  t1.join();  t2.join();  t3.join();

  std::cout << "pi is ~ " << pi << std::endl;

  return 0;
}
```

```
$ ./a.out
```

Mutex

```cpp
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(int begin, int end, double h) {
  for (int i = begin; i < end; ++i) {
    pi_mutex.lock();
    pi += (h*4.0) / (1.0 + (i*h*i*h));
    pi_mutex.unlock();
  }
}
```

Mutex

```
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(int begin, int end, double h) {
  pi_mutex.lock();
  for (int i = begin; i < end; ++i) {
    pi += (h*4.0) / (1.0 + (i*h*i*h));
  }
  pi_mutex.unlock();
}
```

Locking and unlocking at every function call

```
$ time ./a.out    # with race
```

Fast! But wrong!

Fast! And right!

Mutex

Locking and unlocking at every function call

```cpp
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(unsigned long begin, unsigned long end, double h) {
  pi_mutex.lock();
  for (unsigned long i = begin; i < end; ++i) {
    pi += (h*4.0) / (1.0 + (i*h*i*h));
  }
  pi_mutex.unlock();
}
```

```
$ time ./a.out 1000000000 # sequential
pi is ~ 3.14159265458978
2.013u 0.003s 0:02.01 100.0%
```

Why not?

Right! And fast!
But not scaling!

Mutex

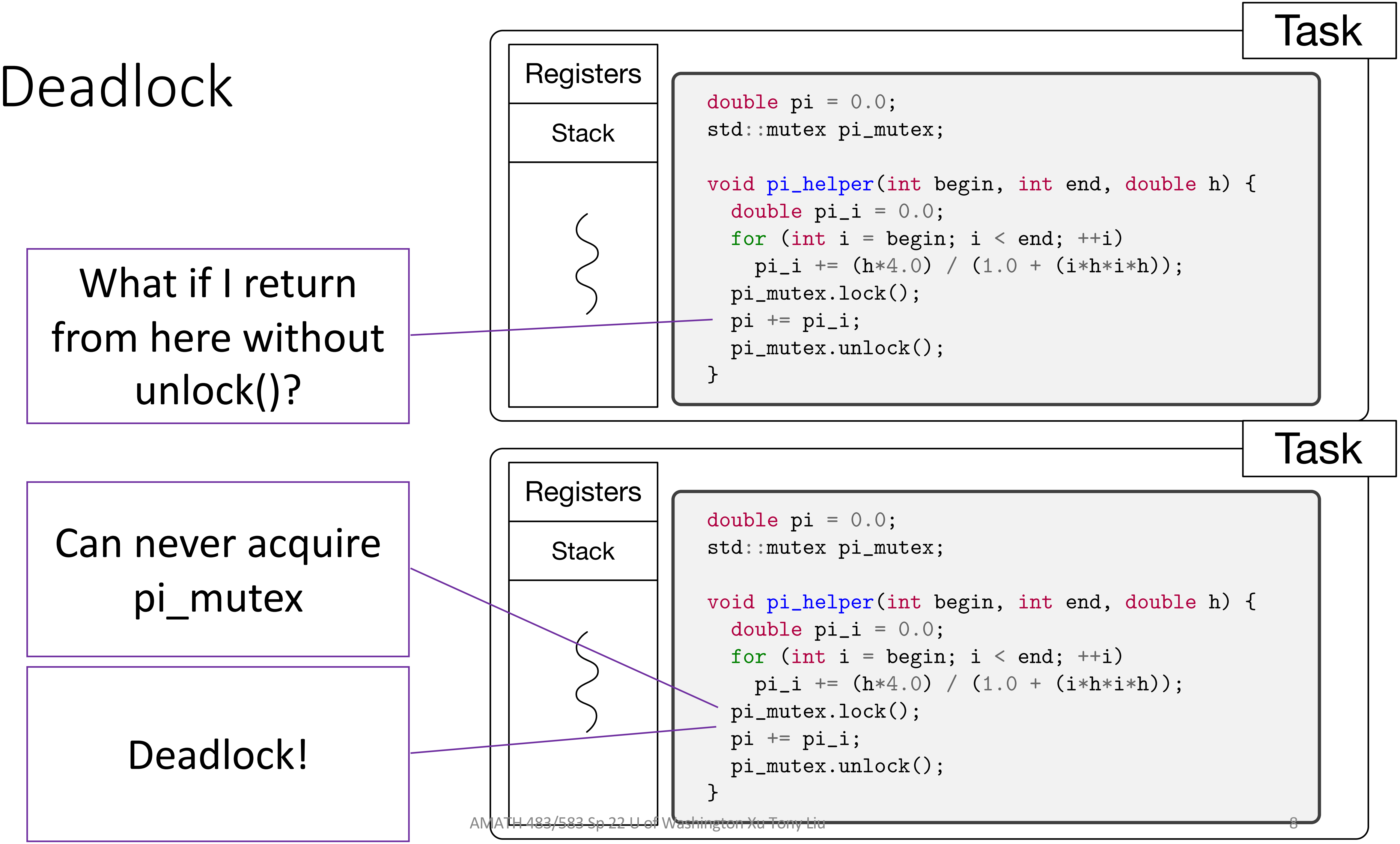
Locking and unlocking at every function call

```
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(unsigned long begin, unsigned long end, double h) {
  pi_mutex.lock();
  for (unsigned long i = begin; i < end; ++i) {
    pi += (h*4.0) / (1.0 + (i*h*i*h));
  }
  pi_mutex.unlock();
}
```

```
$ time ./a.out 1000000000 # sequential
pi is ~ 3.14159265458978
2.013u 0.003s 0:02.01 100.0%
```

Can multiple threads run this in parallel? (or even concurrently?)

Deadlock



```
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(int begin, int end, double h) {
  double pi_i = 0.0;
  for (int i = begin; i < end; ++i)
    pi_i += (h*4.0) / (1.0 + (i*h*i*h));
  pi_mutex.lock();
  pi += pi_i;
  pi_mutex.unlock();
}
```

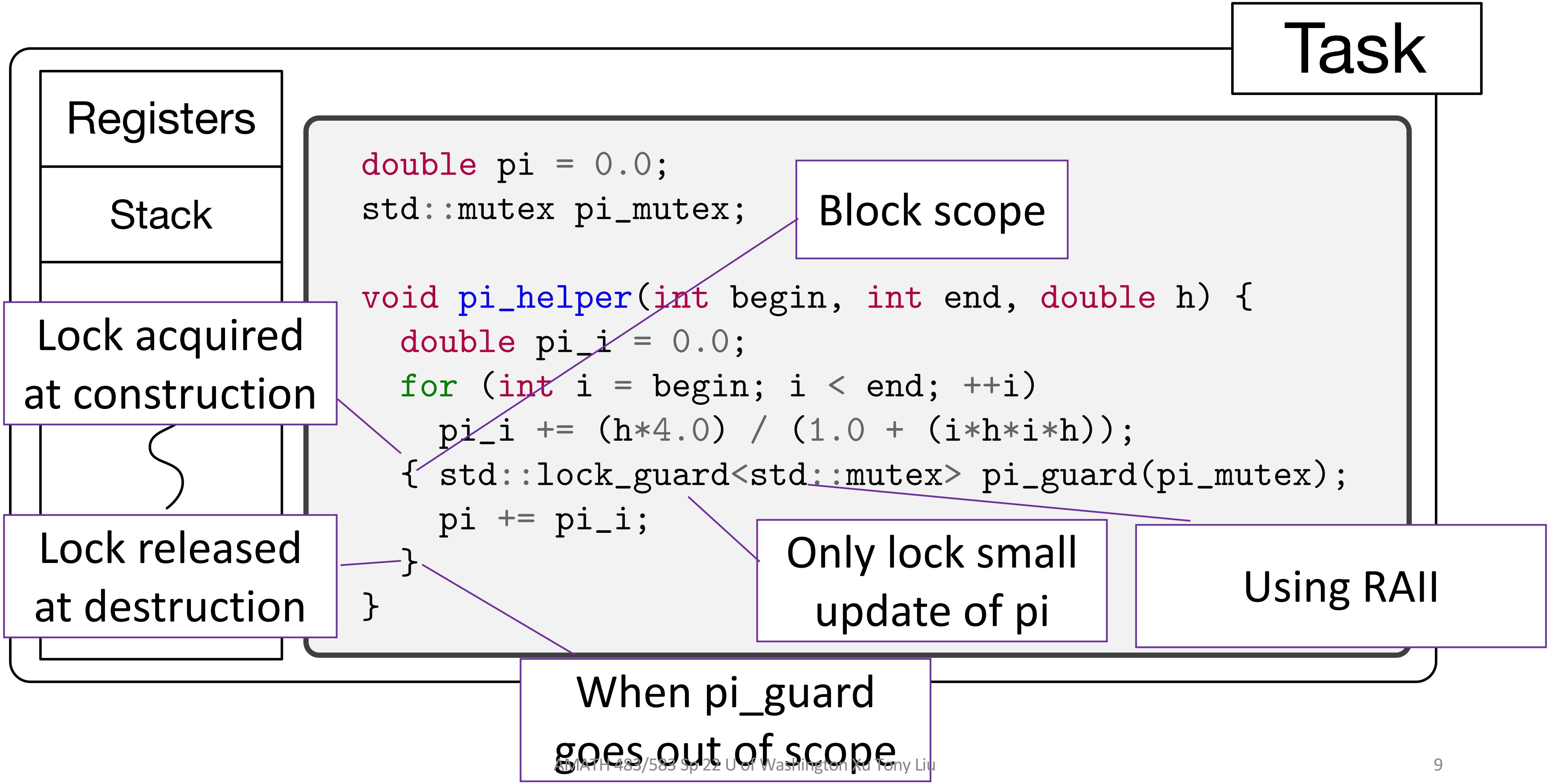
What if I return from here without unlock()?

```
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(int begin, int end, double h) {
  double pi_i = 0.0;
  for (int i = begin; i < end; ++i)
    pi_i += (h*4.0) / (1.0 + (i*h*i*h));
  pi_mutex.lock();
  pi += pi_i;
  pi_mutex.unlock();
}
```

Can never acquire pi_mutex

Deadlock!

Lock Guard



```
double pi = 0.0;
std::mutex pi_mutex;

void pi_helper(int begin, int end, double h) {
  double pi_i = 0.0;
  for (int i = begin; i < end; ++i)
    pi_i += (h*4.0) / (1.0 + (i*h*i*h));
  { std::lock_guard<std::mutex> pi_guard(pi_mutex);
    pi += pi_i;
  }
}
```

Block scope

Lock acquired at construction

Lock released at destruction

Only lock small update of pi

Using RAII

When pi_guard goes out of scope

Numerical Quadrature (Tasks)

```
int main(int argc, char *argv[])
{
  unsigned long intervals  = 1024*1024, num_blocks  = 128, blocksize = intervals / num_blocks;
  double h = 1.0 / (double) intervals;

  std::vector<std::future<double> > partial_sums;

  for (unsigned long k = 0; k < num_blocks; ++k) {
    partial_sums.push_back(std::async(partial_pi, k*blocksize, (k+1)*blocksize, h));
  }

  double pi = 0.0;
  for (unsigned long k = 0; k < num_blocks; ++k) {
    pi += h*partial_sums[k].get();
  }
  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

Promise a double

Vector of futures

Launch tasks: each computes a partial sum

Cash in the IOUs

Launching async()

```
int main(int argc, char* argv[]) {
  unsigned long intervals   = 1024 * 1024;
  unsigned long num_blocks  = 1;
  double        h           = 1.0 / (double)intervals;
  unsigned long blocksize   = intervals / num_blocks;

  std::vector<std::future<double>> partial_sums;

  for (unsigned long k = 0; k < num_blocks; ++k)
    partial_sums.push_back(
      std::async(std::launch::async,
        partial_pi, k * blocksize, (k + 1) * blocksize, h));

  for (unsigned long k = 0; k < num_blocks; ++k)
    pi += h * partial_sums[k].get();

  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

Run right away

Results will be here

Bonnie and Clyde Redux

```
int bank_balance = 300;

static std::mutex atm_mutex;
static std::mutex msg_mutex;

void withdraw(const string& msg, int amou
  std::lock(atm_mutex, msg_mutex);
  std::lock_guard<std::mutex> message_lock(msg_mutex, std::adopt_lock);

  cout << msg << " withdraws " << to_string(amount) << endl;

  std::lock_guard<std::mutex> account_lock(atm_mutex, std::adopt_lock);

  bank_balance -= amount;
}
```

Mutexes

Avoid deadlock

Lock two mutexes at once

std::atomic

Bank balance is an indivisible type

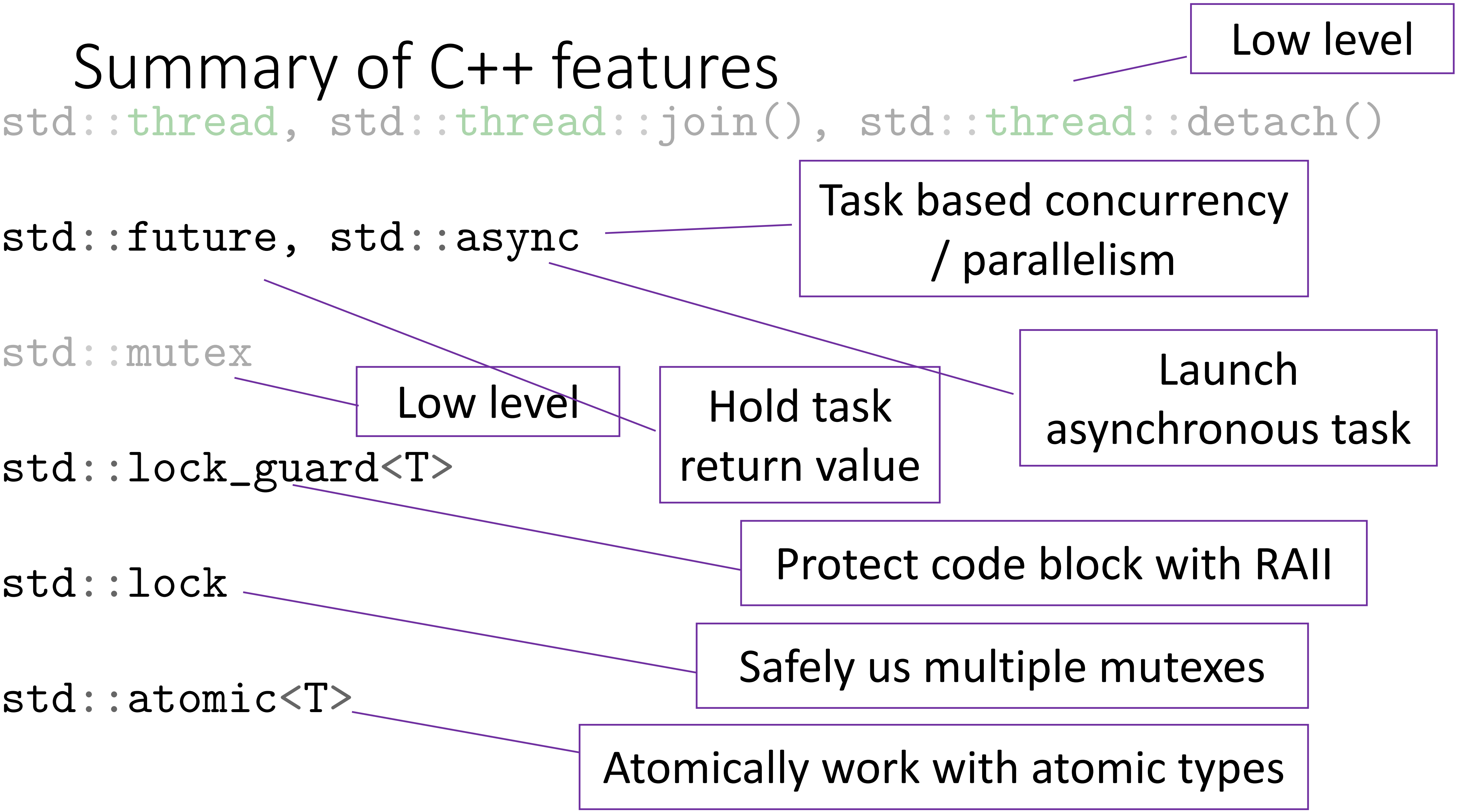
```
atomic<int>        bank_balance(300
static std::mutex msg_mutex;

void withdraw(const string& msg, int amount) {

  { std::lock_guard<std::mutex> message_lock(msg_mutex);
    cout << msg << " withdraws " << to_string(amount) << endl;
  }

  bank_balance -= amount;
}
```

NB!! no longer equivalent to
bank_balance = bank_balance – amount;

operator+=(), e.g.

Certain operators are guaranteed to be atomic

Summary of C++ features



`std::thread, std::thread::join(), std::thread::detach()`

Low level

`std::future, std::async`

Task based concurrency / parallelism

Hold task return value

Launch asynchronous task

`std::mutex`

Low level

`std::lock_guard<T>`

Protect code block with RAII

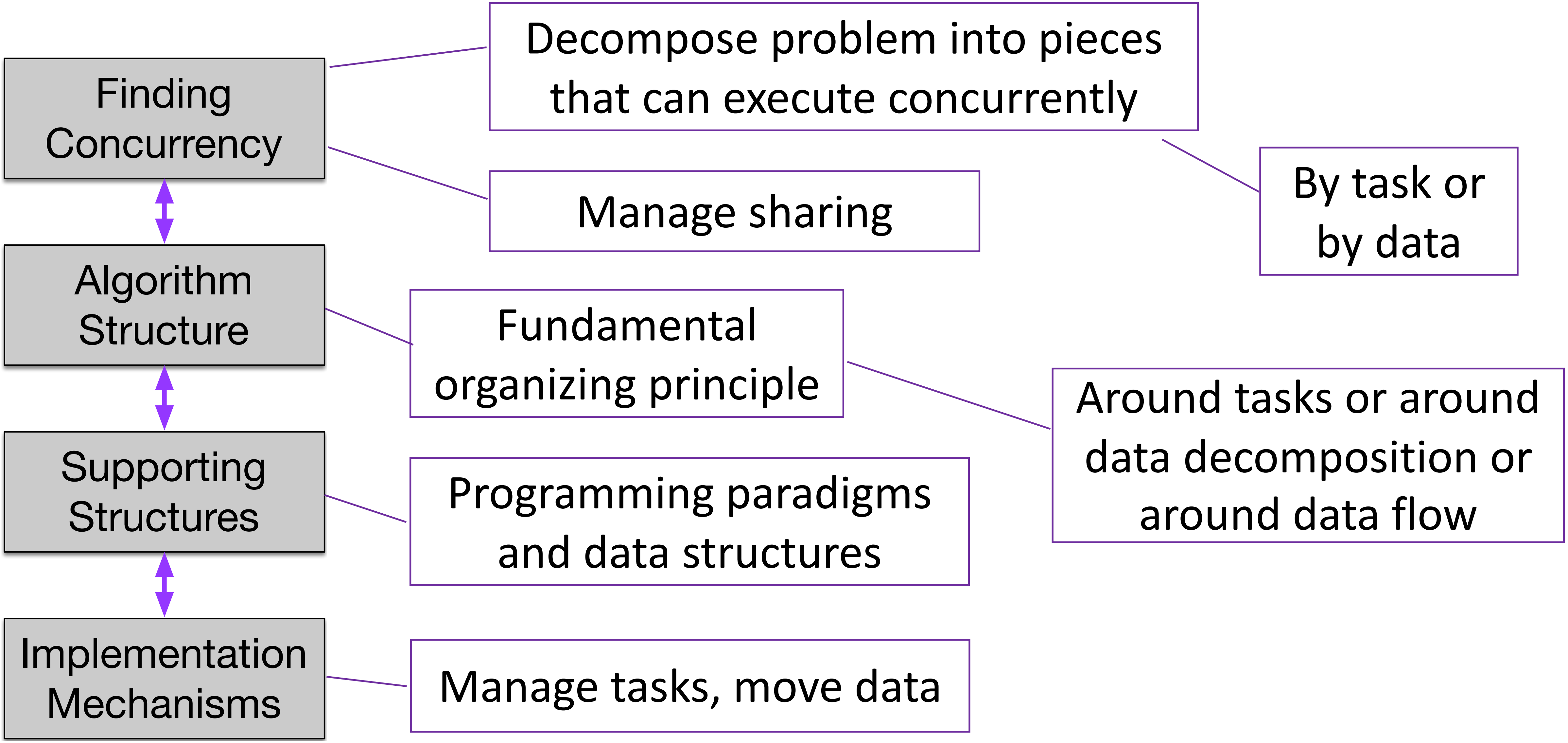
`std::lock`

Safely us multiple mutexes

`std::atomic<T>`

Atomically work with atomic types

Parallelization Strategy



Finding Concurrency

Algorithm Structure

Supporting Structures

Implementation Mechanisms

Decompose problem into pieces that can execute concurrently

By task or by data

Manage sharing

Fundamental organizing principle

Around tasks or around data decomposition or around data flow

Programming paradigms and data structures

Manage tasks, move data

Timothy Mattson, Beverly Sanders, and Berna Massingill. 2004. *Patterns for Parallel Programming*(First ed.). Addison-Wesley Professional.

Two Norm Function (Sequential)

```
double two_norm(const Vector& x) {
  double sum = 0.0;
  for (size_t i = 0; i < x.num_rows(); ++i) {
    sum += x(i) * x(i);
  }
  return std::sqrt(sum);
}
```

Partitioned Vector

```
class PartitionedVector {
public:
  PartitionedVector(size_t M) : num_rows_(M), storage_(num_rows_) {}

        double& operator()(size_t i)       { return storage_[i]; }
  const double& operator()(size_t i) const { return storage_[i]; }

  size_t num_rows() const { return num_rows_; }

  void partition_by_rows(size_t parts) {
    size_t xsize = num_rows_ / parts;
    partitions_.resize(parts+1);
    std::fill(partitions_.begin()+1, partitions_.end(), xsize);
    std::partial_sum(partitions_.begin(), partitions_.end(), partitions_.begin());
  }


private:
  size_t              num_rows_;
  std::vector<double> storage_;
public:
  std::vector<size_t> partitions_;
};
```

Two Norm v.1

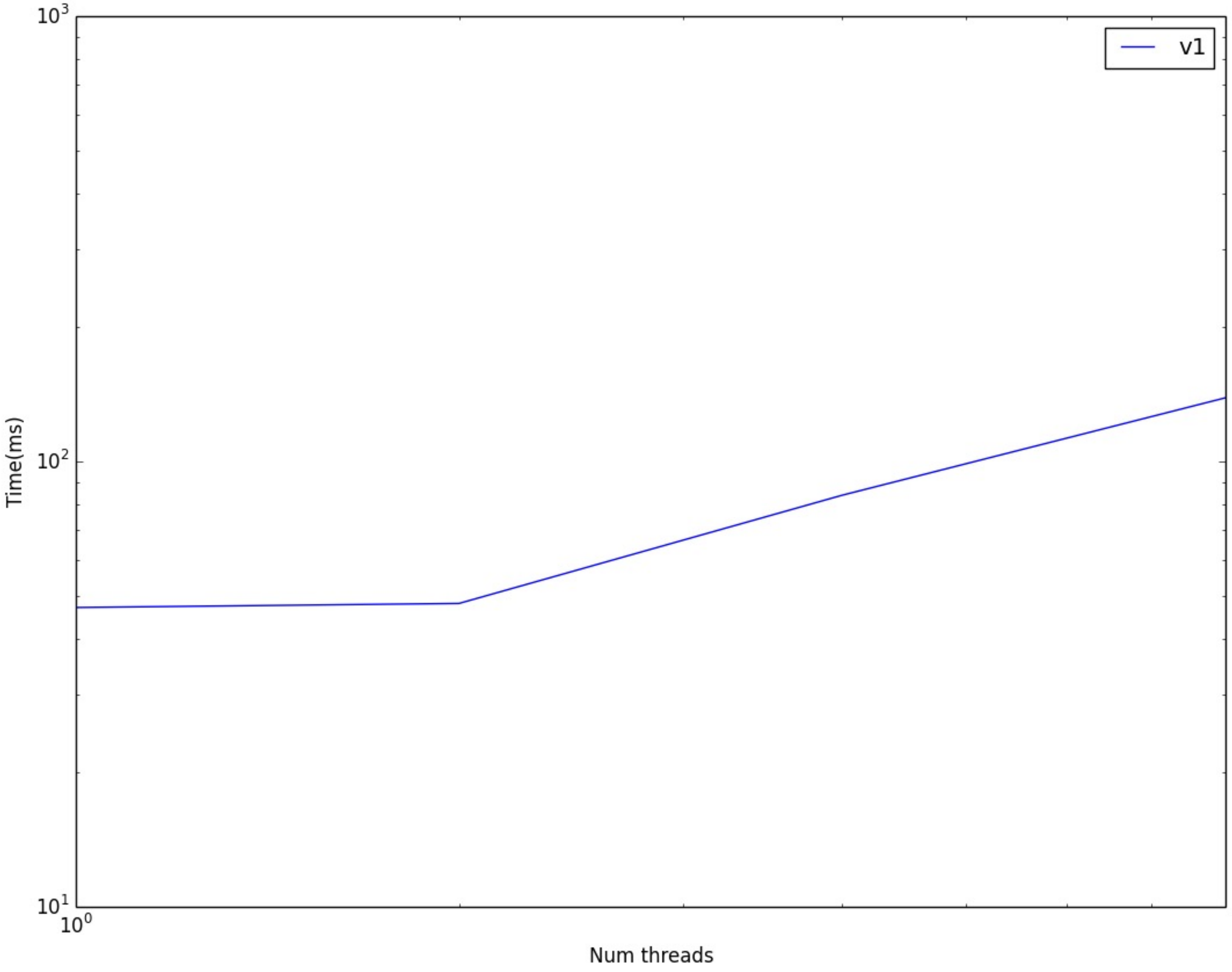
```
double two_norm_part(const PartitionedVector& x, size_t p) {
  double sum = 0.0;
  for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
    sum += x(i) * x(i);
  }
  return sum;
}



double two_norm_px(const PartitionedVector& x) {
  std::vector<std::future<double>> futures_;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    futures_.push_back(std::async(std::launch::async, two_norm_part, x, p));
  }

  double sum = 0.0;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    sum += futures_[p].get();
  }
  return std::sqrt(sum);
}
```

Pass-by-ref or pass-by-value?

Timing

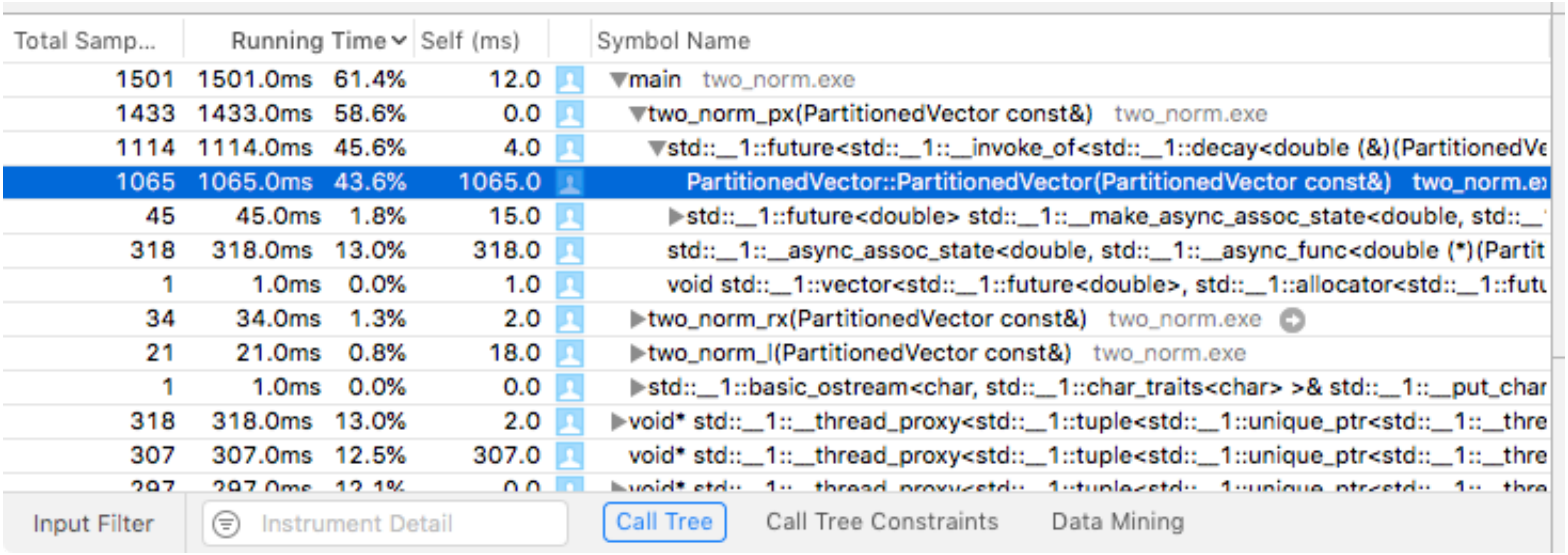
```
for (size_t num_threads = 1; num_threads <= 8; num_threads*=2) {
    x.partition_by_rows(num_threads);

    DEF_TIMER(two_norm_rx);
    START_TIMER(two_norm_rx);
    for (size_t i = 0; i < trips; ++i) {
      b += two_norm_rx(x);
    }
    STOP_TIMER(two_norm_rx);
```

Results



→ 8

What Happened?



Total Samp...	Running Time		Self (ms)	Symbol Name
1501	1501.0ms	61.4%	12.0	▼main two_norm.exe
1433	1433.0ms	58.6%	0.0	▼two_norm_px(PartitionedVector const&) two_norm.exe
1114	1114.0ms	45.6%	4.0	▼std::__1::future<std::__1::__invoke_of<std::__1::decay<double (&)(PartitionedVe
1065	1065.0ms	43.6%	1065.0	PartitionedVector::PartitionedVector(PartitionedVector const&) two_norm.ex
45	45.0ms	1.8%	15.0	▶std::__1::future<double> std::__1::__make_async_assoc_state<double, std::__
318	318.0ms	13.0%	318.0	std::__1::__async_assoc_state<double, std::__1::__async_func<double (*)(Partit
1	1.0ms	0.0%	1.0	void std::__1::vector<std::__1::future<double>, std::__1::allocator<std::__1::futu
34	34.0ms	1.3%	2.0	▶two_norm_rx(PartitionedVector const&) two_norm.exe
21	21.0ms	0.8%	18.0	▶two_norm_l(PartitionedVector const&) two_norm.exe
1	1.0ms	0.0%	0.0	▶std::__1::basic_ostream<char, std::__1::char_traits<char> >& std::__1::__put_char
318	318.0ms	13.0%	2.0	▶void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thre
307	307.0ms	12.5%	307.0	void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thre
297	297.0ms	12.1%	0.0	▶void* std::__1::__thread_proxy<std::__1::tuple<std::__1::unique_ptr<std::__1::__thre

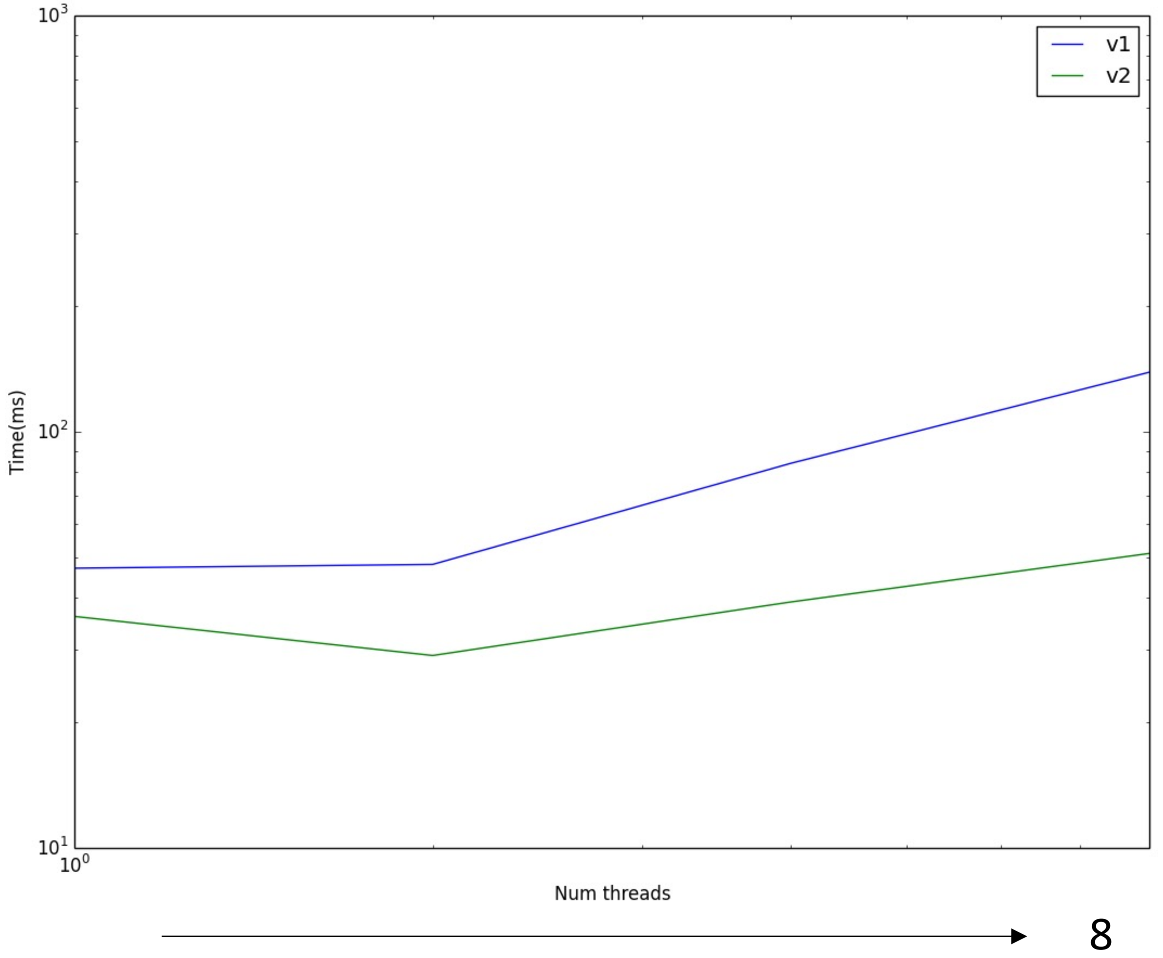
Two Norm v.2

```
double two_norm_part(const PartitionedVector& x, size_t p) {
  double sum = 0.0;
  for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
    sum += x(i) * x(i);
  }
  return sum;
}

double two_norm_rx(const PartitionedVector& x) {
  std::vector<std::future<double>> futures_;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    futures_.push_back(std::async(std::launch::async, two_norm_part, std::cref(x), p));
  }

  double sum = 0.0;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    sum += futures_[p].get();
  }
  return std::sqrt(sum);
}
```

std::async do NOT check how the argument will eventually be used

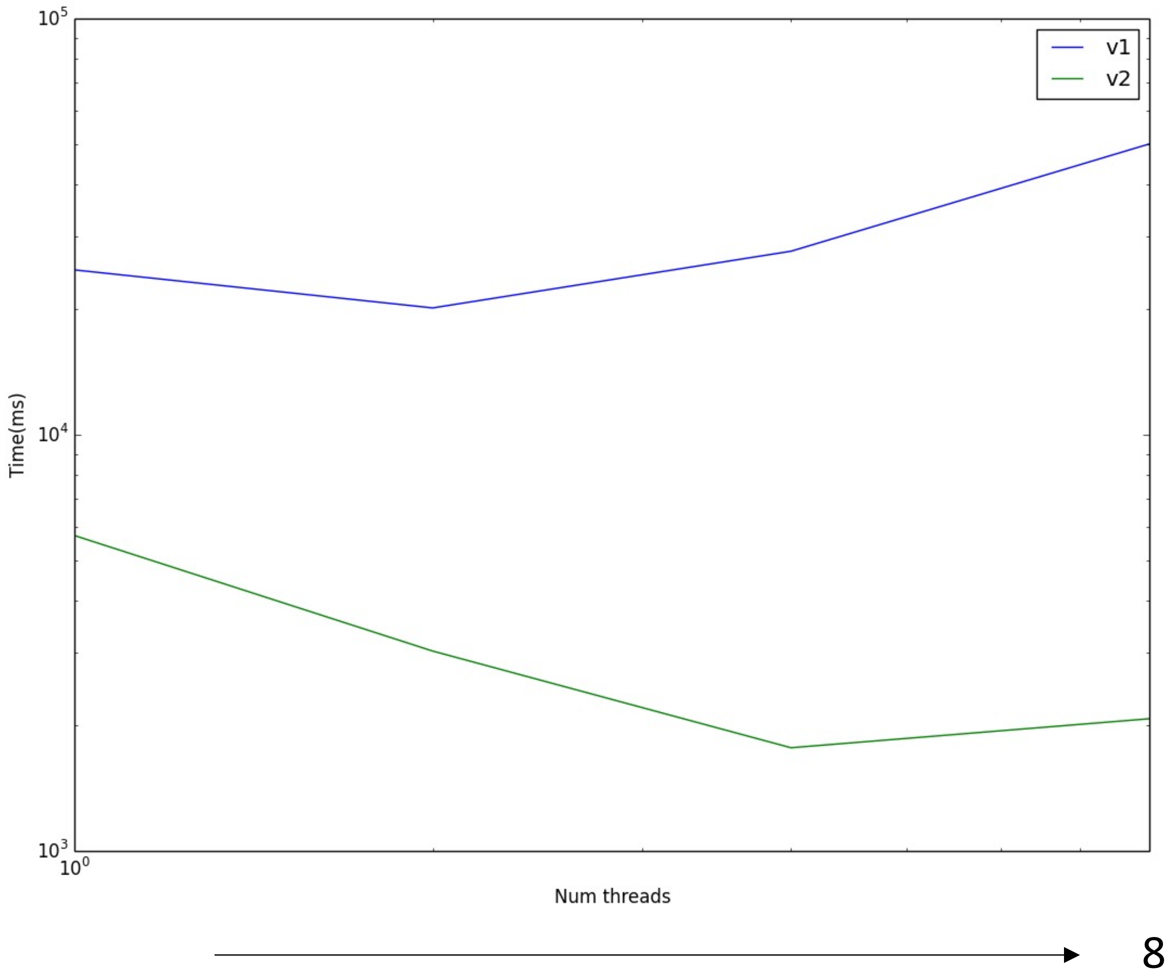
Explicitly pass-by-ref

Results v.2



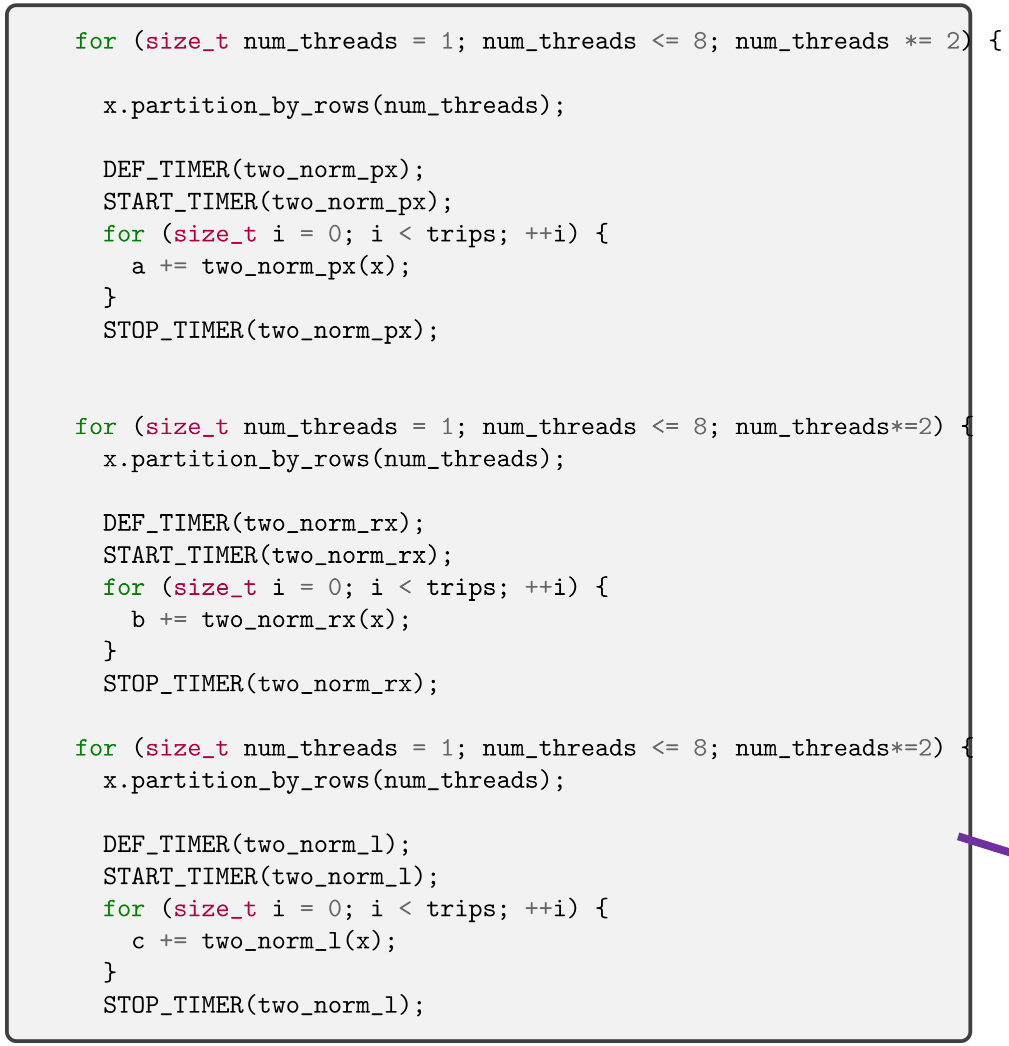
With small vector

Results v.2



With large vector

Timing all Three Norms



```
for (size_t num_threads = 1; num_threads <= 8; num_threads *= 2) {

  x.partition_by_rows(num_threads);

  DEF_TIMER(two_norm_px);
  START_TIMER(two_norm_px);
  for (size_t i = 0; i < trips; ++i) {
    a += two_norm_px(x);
  }
  STOP_TIMER(two_norm_px);


for (size_t num_threads = 1; num_threads <= 8; num_threads*=2) {
  x.partition_by_rows(num_threads);

  DEF_TIMER(two_norm_rx);
  START_TIMER(two_norm_rx);
  for (size_t i = 0; i < trips; ++i) {
    b += two_norm_rx(x);
  }
  STOP_TIMER(two_norm_rx);

for (size_t num_threads = 1; num_threads <= 8; num_threads*=2) {
  x.partition_by_rows(num_threads);

  DEF_TIMER(two_norm_l);
  START_TIMER(two_norm_l);
  for (size_t i = 0; i < trips; ++i) {
    c += two_norm_l(x);
  }
  STOP_TIMER(two_norm_l);
```

These are all the same

Functions as Values

```
void benchmark(const PartitionedVector& x) {
  for (size_t num_threads = 1; num_threads <= 8;              2) {

    x.partition_by_rows(num_threads);

    DEF_TIMER(two_norm_px);
    START_TIMER(two_norm_px);
    for (size_t i = 0; i < trips; ++i) {
      a += <something>(x);
    }
    STOP_TIMER(two_norm_px)
}
```

We want to pass in something

That we call like a function

Double bonus: It just needs an operator()()

Let’s not get carried away

Functions as Values

```
void bench(std::function<double (PartitionedVector&)> two_norm_f,
           PartitionedVector& x) {

  double a = 0;
  for (size_t num_threads = 1; num_threads <= 8; nu                 {

    x.partition_by_rows(num_threads);

    DEF_TIMER(two_norm_px);
    START_TIMER(two_norm_px);
    for (size_t i = 0; i < trips; ++i) {
      a += two_norm_f(std::ref(x));
    }
    STOP_TIMER(two_norm_px);
  }
}
```

Is a function

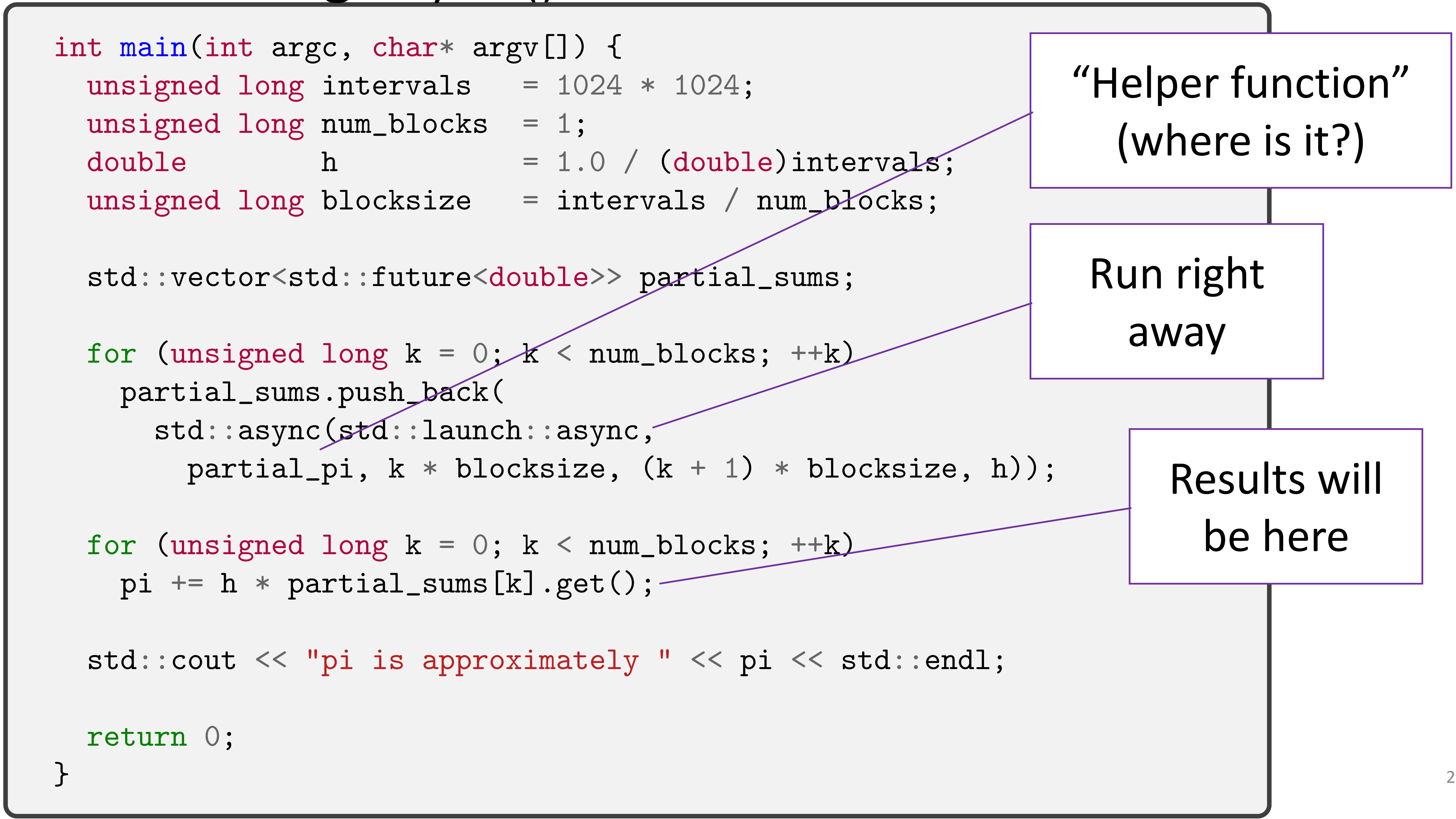
Parameter f

That returns void

Two Norm v.2

```cpp
double two_norm_part(const PartitionedVector& x, size_t p) {
  double sum = 0.0;
  for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
    sum += x(i) * x(i);
  }
  return sum;
}

double two_norm_rx(const PartitionedVector& x) {
  std::vector<std::future<double>> futures_;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    futures_.push_back(std::async(std::launch::async, two_norm_part, std::cref(x), p));
  }

  double sum = 0.0;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    sum += futures_[p].get();
  }
  return std::sqrt(sum);
}
```

Launching async()



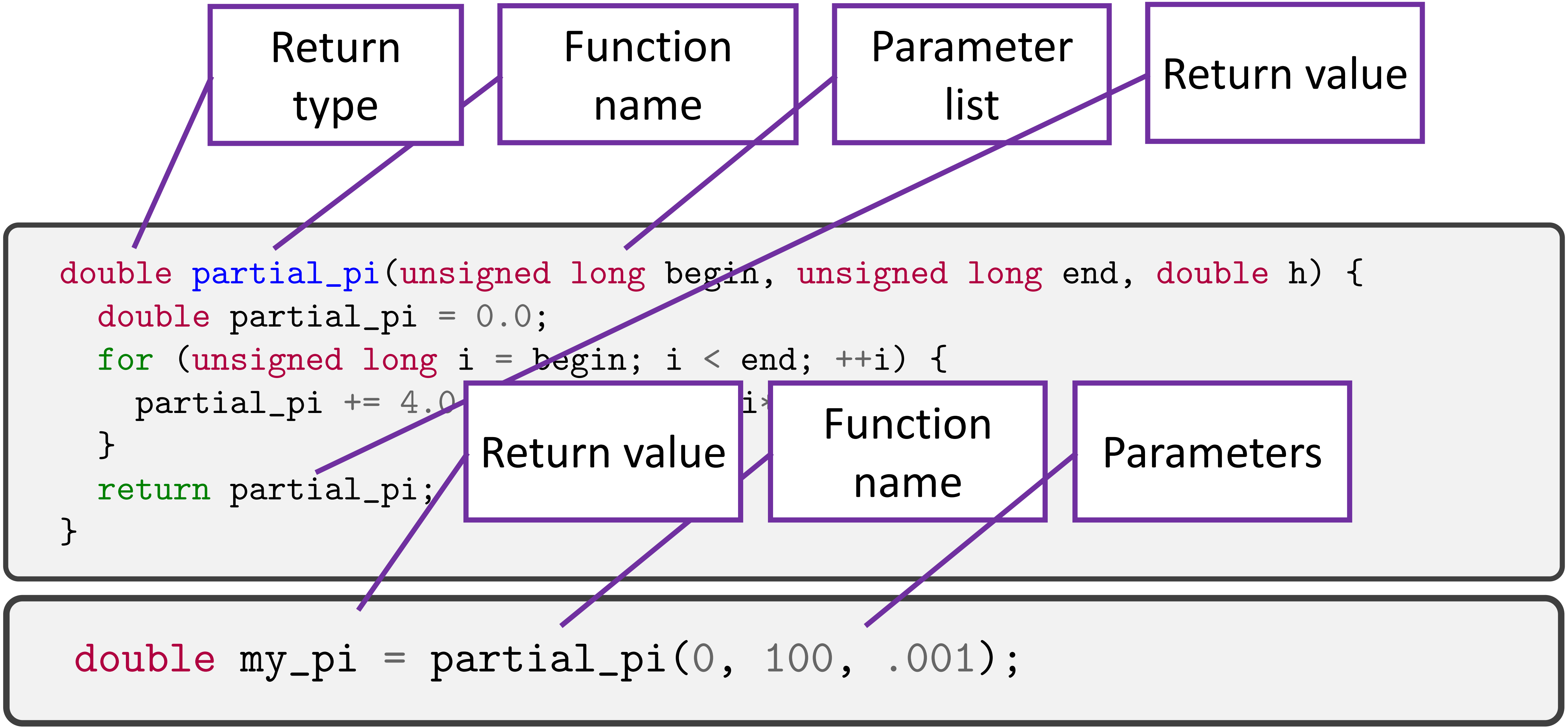
```
int main(int argc, char* argv[]) {
  unsigned long intervals   = 1024 * 1024;
  unsigned long num_blocks  = 1;
  double        h           = 1.0 / (double)intervals;
  unsigned long blocksize   = intervals / num_blocks;

  std::vector<std::future<double>> partial_sums;

  for (unsigned long k = 0; k < num_blocks; ++k)
    partial_sums.push_back(
      std::async(std::launch::async,
        partial_pi, k * blocksize, (k + 1) * blocksize, h));

  for (unsigned long k = 0; k < num_blocks; ++k)
    pi += h * partial_sums[k].get();

  std::cout << "pi is approximately " << pi << std::endl;

  return 0;
}
```

“Helper function”
(where is it?)

Run right
away

Results will
be here

Named function



Return type

Function name

Parameter list

Return value

```
double partial_pi(unsigned long begin, unsigned long end, double h) {
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i) {
    partial_pi += 4.0                 i*
  }
  return partial_pi;
}
```

Return value

Function name

Parameters

```
double my_pi = partial_pi(0, 100, .001);
```

Named functions

```
double partial_pi(unsigned long begin, unsig                double h) {
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}
```

But what is this really?

Function name

Parameters

```
partial_sums.push_back(
  std::async(std::launch::async,
    partial_pi, k * blocksize, (k + 1) * blocksize, h));
```

Named variables

Variable name

Variable value

```
double pi = 3.14;

double sqrtpi_1 = sqrt583(pi);

double sqrtpi_2 = sqrt583(3.14);
```

Call with variable name

Value will be looked up

And then sqrt583 will be called

Call with value

Function will be called with same thing as before

Named functions

Function name

```
double partial_pi(unsigned long begin, unsigned long en
  double partial_pi = 0.0;
  for (unsigned long i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}
```

Can I call std::async directly with the value of partial_pi

(yes)

Value will be looked up

Call with function name

And then std::async will be called

```
partial_sums.push_back(
  std::async(std::launch::async,
    partial_pi, k * blocksize, (k + 1) * blocksize, h));
```

Name this famous person

Alonzo Church (June 14, 1903 – August 11, 1995) was an American mathematician and logician who made major contributions to mathematical logic and the foundations of theoretical computer science. He is best known for the ***lambda calculus***, Church–Turing thesis, proving the undecidability of the Entscheidungs problem, Frege–Church ontology, and the Church–Rosser theorem.

Various formalisms for computing

Gottlog Frege

Alan Turing

John Barkley Rosser

Anonymous Function

Anonymous function or lambda function or lambda

  - A function definition that is not bound to an identifier
  - Invented by Alonzo Church in 1936
  - Became a feature in Lisp in 1958
  - Supported by more and more modern programming languages
  - Added to C++ since C++11

- Lambda in C++
  - An unnamed function object capable of capturing variables in scope

Lambda: Anonymous functions

```
int main(int argc, char* argv[]) {
  unsigned long intervals    = 1024 * 1024;
  unsigned long num_blocks   = 1;
  double        h            = 1.0 / (double)intervals;
  unsigned long blocksize    = intervals / num_blocks;

  std::vector<std::future<double>> partial_sums;

  for (unsigned long k = 0; k < num_blocks; ++k) {
    partial_sums.push_back(std::async(std::launch::async, [&]() -> double {
      double partial_pi = 0.0;
      for (unsigned long i = k * blocksize; i < (k + 1) * blocksize; ++i) {
        partial_pi += 4.0 / (1.0 + (i * h * i * h));
      }
      return partial_pi;
    }));
  }

  double pi = 0.0;
  for (unsigned long k = 0; k < num_blocks; ++k) {
    pi += h * partial_sums[k].get();
  }
  std::cout << "pi is approximately " << std::setprecision(15) << pi << std::endl;

  return 0;
}
```

Lambda: Anonymous functions

```
for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [](size_t begin, size_t end, double h) -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = begin; i < end; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
}
```

Value of partial_pi

Two Norm v.3

```
double two_norm_l(const PartitionedVector& x) {
  std::vector<std::future<double>> futures_;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    futures_.emplace_back(std::async(std::launch::async, [&](size_t p) {
    double sum = 0.0;
    for (size_t i = x.partitions_[p]; i < x.partitions_[p+1]; ++i) {
      sum += x(i) * x(i);
    }
    return sum;
  }, p));

  }

  double sum = 0.0;
  for (size_t p = 0; p < x.partitions_.size()-1; ++p) {
    sum += futures_[p].get();
  }
  return std::sqrt(sum);
}
```

Used to be two_norm_part

lambda

Before

```
double partial_pi(size_t begin, size_t end, double h)
{
  double partial_pi = 0.0;
  for (size_t i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}
```

After

```cpp
auto partial_pi(size_t begin, size_t end, double h) -> double
{
  double partial_pi = 0.0;
  for (size_t i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}
```

Before

```
auto partial_pi(size_t begin, size_t end, double h) -> double
{
  double partial_pi = 0.0;
  for (size_t i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
}
```

After

```
auto partial_pi = [](size_t begin, size_t end, double h) -> double
{
  double partial_pi = 0.0;
  for (size_t i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
};
```

Function values

“Lambda” (this is a function value)

Function parameters

```
auto partial_pi = [](size_t begin, size_t end, double h) -> double
{
  double partial_pi = 0.0;
  for (size_t i = begin; i < end; ++i) {
    partial_pi += 4.0 / (1.0 + (i*h*i*h));
  }
  return partial_pi;
};
```

Return type

Lambda can be assigned to an auto type

Return value

What is the value of partial_pi?

Before

```
    (std::async(std::launch::async,
                partial_pi,







                   k * blocksize, (k + 1) * blocksize, h
                ));
```

After

```
(std::async(std::launch::async,
            [](size_t begin, size_t end, double h) -> double
            {
              double partial_pi = 0.0;
              for (size_t i = begin; i < end; ++i) {
                partial_pi += 4.0 / (1.0 + (i*h*i*h));
              }
              return partial_pi;
            }, k * blocksize, (k + 1) * blocksize, h
            ));
```

Before

```
(std::async(std::launch::async,
            partial_pi,




               k * blocksize, (k + 1) * blocksize, h
            ));
```

Function name

After

Function value

async “sees” the same thing

```
(std::async(std::launch::async,
            [](size_t begin, size_t end, double h) -> double
            {
              double partial_pi = 0.0;
              for (size_t i = begin; i < end; ++i) {
                partial_pi += 4.0 / (1.0 + (i*h*i*h));
              }
              return partial_pi;
            }, k * blocksize, (k + 1) * blocksize, h
            ));
```

All together

```
int main(int argc, char* argv[]) {
  size_t intervals   = 1024 * 1024;
  size_t num_blocks  = 1;
  double        h           = 1.0 / (double)intervals;
  size_t blocksize   = intervals / num_blocks;

  std::vector<std::future<double>> partial_sums;

  for (size_t k = 0; k < num_blocks; ++k) {
    partial_sums.push_back
      (std::async(std::launch::async,
                  [](size_t begin, size_t end, double h) -> double
                  {
                    double partial_pi = 0.0;
                    for (size_t i = begin; i < end; ++i) {
                      partial_pi += 4.0 / (1.0 + (i*h*i*h));
                    }
                    return partial_pi;
                   , k * blocksize, (k + 1) * blocksize, h
                  ));
  }

  double pi = 0.0;
  for (size_t k = 0; k < num_blocks; ++k) {
    pi += h * partial_sums[k].get();
  }
  std::cout << "pi is approximately " << std::setprecision(15) <<
 ↪  pi << std::endl;

  return 0;
}
```

All together zoomed

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h          = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<double>> partial_sums;

for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [](size_t begin, size_t end, double h) -> double
                {
                  double partial_pi = 0.0;
                     r (size_t i = begin; i < end; ++i) {
                      artial_pi += 4.0 / (1.0 + (i*h*i*h));

                     turn partial_pi;
                 , k * blocksize, (k + 1) * blocksize, h
                ));
```

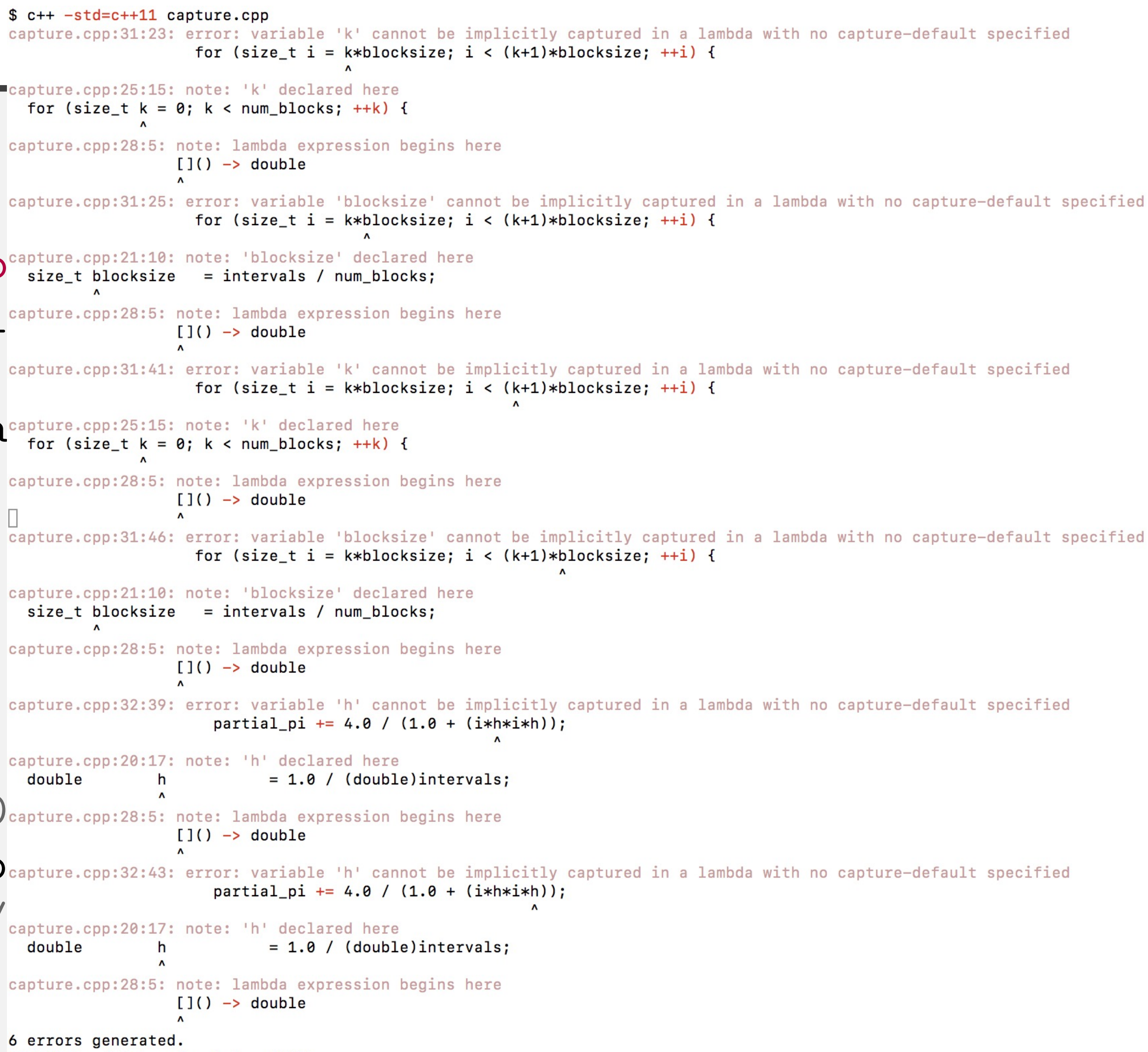
Function parameters

Why can't we use k, blocksize, and h directly?

Passed parameters

Capture

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (doub
size_t blocksize   = intervals / num_bl

std::vector<std::future<double>> partia

for (size_t k = 0; k < num_blocks; ++k)
  partial_sums.push_back
    (std::async(std::launch::async,
                []() -> double
                {
                  double partial_pi = 0
                  for (size_t i = k*blo
                    partial_pi += 4.0 /
                  }
                  return partial_pi;
                }
                ));
```



```
$ c++ -std=c++11 capture.cpp
capture.cpp:31:23: error: variable 'k' cannot be implicitly captured in a lambda with no capture-default specified
                    for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                                    ^
capture.cpp:25:15: note: 'k' declared here
  for (size_t k = 0; k < num_blocks; ++k) {
              ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
capture.cpp:31:25: error: variable 'blocksize' cannot be implicitly captured in a lambda with no capture-default specified
                    for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                                      ^
capture.cpp:21:10: note: 'blocksize' declared here
  size_t blocksize   = intervals / num_blocks;
         ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
capture.cpp:31:41: error: variable 'k' cannot be implicitly captured in a lambda with no capture-default specified
                    for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                                                      ^
capture.cpp:25:15: note: 'k' declared here
  for (size_t k = 0; k < num_blocks; ++k) {
              ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
capture.cpp:31:46: error: variable 'blocksize' cannot be implicitly captured in a lambda with no capture-default specified
                    for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                                                           ^
capture.cpp:21:10: note: 'blocksize' declared here
  size_t blocksize   = intervals / num_blocks;
         ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
capture.cpp:32:39: error: variable 'h' cannot be implicitly captured in a lambda with no capture-default specified
                      partial_pi += 4.0 / (1.0 + (i*h*i*h));
                                                    ^
capture.cpp:20:17: note: 'h' declared here
  double        h           = 1.0 / (double)intervals;
                ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
capture.cpp:32:43: error: variable 'h' cannot be implicitly captured in a lambda with no capture-default specified
                      partial_pi += 4.0 / (1.0 + (i*h*i*h));
                                                        ^
capture.cpp:20:17: note: 'h' declared here
  double        h           = 1.0 / (double)intervals;
                ^
capture.cpp:28:5: note: lambda expression begins here
                  []() -> double
                  ^
6 errors generated.
```

Before

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<double>> partial_sums;

for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [=]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

After

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<double>> partial_sums;

for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [&]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

After after

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<double>> partial_sums;

for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [=]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

After after after

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<double>> partial_sums;

for (size_t k = 0; k < num_blocks; ++k) {
  partial_sums.push_back
    (std::async(std::launch::async,
                [k, blocksize, &h]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

Capture all by reference

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<doub

for (size_t k = 0; k < num_b
  partial_sums.push_back
    (std::async(std::launch:
                [&]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

Capture all by reference

Capture all by value

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<doub

for (size_t k = 0; k < num_b
  partial_sums.push_back
    (std::async(std::launch:
                [=]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

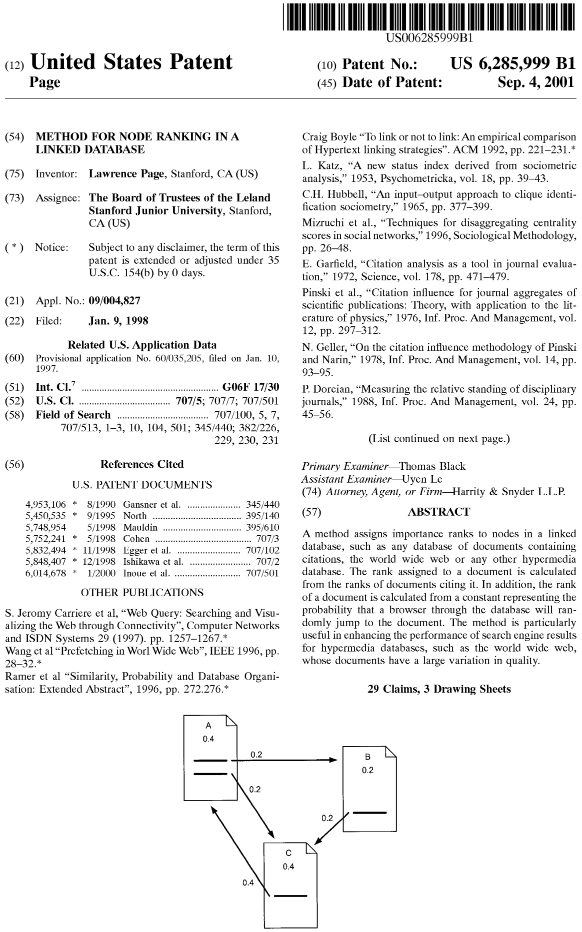
Capture all by value

Capture some by value, some by reference

```
size_t intervals   = 1024 * 1024;
size_t num_blocks  = 1;
double        h           = 1.0 / (double)intervals;
size_t blocksize   = intervals / num_blocks;

std::vector<std::future<doub

for (size_t k = 0; k < num_b
  partial_sums.push_back
    (std::async(std::launch:
                [k, blocksize, &h]() -> double
                {
                  double partial_pi = 0.0;
                  for (size_t i = k*blocksize; i < (k+1)*blocksize; ++i) {
                    partial_pi += 4.0 / (1.0 + (i*h*i*h));
                  }
                  return partial_pi;
                }
                ));
```

Pick and choose

Who Wants to be a Billionaire?



US006285999B1

(12) **United States Patent**
Page

(10) **Patent No.: US 6,285,999 B1**
(45) **Date of Patent: Sep. 4, 2001**

---

(54) **METHOD FOR NODE RANKING IN A LINKED DATABASE**

(75) Inventor: **Lawrence Page**, Stanford, CA (US)

(73) Assignee: **The Board of Trustees of the Leland Stanford Junior University**, Stanford, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/004,827**

(22) Filed: **Jan. 9, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/035,205, filed on Jan. 10, 1997.

(51) **Int. Cl.**^{7} **G06F 17/30**
(52) **U.S. Cl.** **707/5**; 707/7; 707/501
(58) **Field of Search** 707/100, 5, 7, 707/513, 1–3, 10, 104, 501; 345/440; 382/226, 229, 230, 231

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,953,106	*	8/1990	Gansner et al.	345/440
5,450,535	*	9/1995	North	395/140
5,748,954		5/1998	Mauldin	395/610
5,752,241	*	5/1998	Cohen	707/3
5,832,494	*	11/1998	Egger et al.	707/102
5,848,407	*	12/1998	Ishikawa et al.	707/2
6,014,678	*	1/2000	Inoue et al.	707/501

OTHER PUBLICATIONS

S. Jeromy Carriere et al, “Web Query: Searching and Visualizing the Web through Connectivity”, Computer Networks and ISDN Systems 29 (1997). pp. 1257–1267.*

Wang et al “Prefetching in Worl Wide Web”, IEEE 1996, pp. 28–32.*

Ramer et al “Similarity, Probability and Database Organisation: Extended Abstract”, 1996, pp. 272.276.*

Craig Boyle “To link or not to link: An empirical comparison of Hypertext linking strategies”. ACM 1992, pp. 221–231.*

L. Katz, “A new status index derived from sociometric analysis,” 1953, Psychometricka, vol. 18, pp. 39–43.

C.H. Hubbell, “An input–output approach to clique identification sociometry,” 1965, pp. 377–399.

Mizruchi et al., “Techniques for disaggregating centrality scores in social networks,” 1996, Sociological Methodology, pp. 26–48.

E. Garfield, “Citation analysis as a tool in journal evaluation,” 1972, Science, vol. 178, pp. 471–479.

Pinski et al., “Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics,” 1976, Inf. Proc. And Management, vol. 12, pp. 297–312.

N. Geller, “On the citation influence methodology of Pinski and Narin,” 1978, Inf. Proc. And Management, vol. 14, pp. 93–95.

P. Doreian, “Measuring the relative standing of disciplinary journals,” 1988, Inf. Proc. And Management, vol. 24, pp. 45–56.

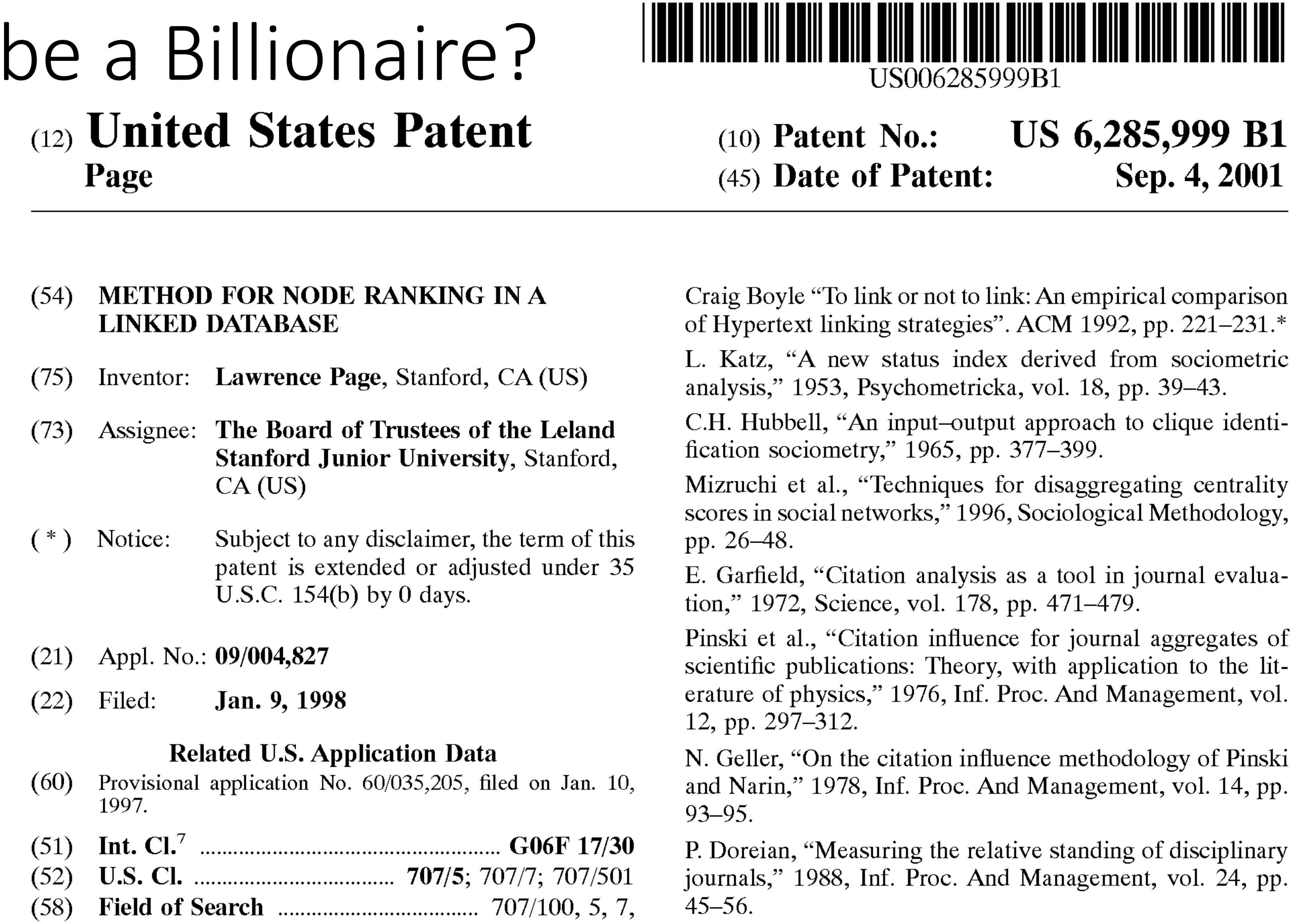
(List continued on next page.)

Primary Examiner—Thomas Black
Assistant Examiner—Uyen Le
(74) *Attorney, Agent, or Firm*—Harrity & Snyder L.L.P.

(57) **ABSTRACT**

A method assigns importance ranks to nodes in a linked database, such as any database of documents containing citations, the world wide web or any other hypermedia database. The rank assigned to a document is calculated from the ranks of documents citing it. In addition, the rank of a document is calculated from a constant representing the probability that a browser through the database will randomly jump to the document. The method is particularly useful in enhancing the performance of search engine results for hypermedia databases, such as the world wide web, whose documents have a large variation in quality.

29 Claims, 3 Drawing Sheets



US006285999B1

(12) **United States Patent**
Page

(10) **Patent No.: US 6,285,999 B1**
(45) **Date of Patent: Sep. 4, 2001**

(54) **METHOD FOR NODE RANKING IN A LINKED DATABASE**

(75) Inventor: **Lawrence Page**, Stanford, CA (US)

(73) Assignee: **The Board of Trustees of the Leland Stanford Junior University**, Stanford, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/004,827**

(22) Filed: **Jan. 9, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/035,205, filed on Jan. 10, 1997.

(51) **Int. Cl.**^{7} **G06F 17/30**
(52) **U.S. Cl.** **707/5**; 707/7; 707/501
(58) **Field of Search** 707/100, 5, 7,

Craig Boyle “To link or not to link: An empirical comparison of Hypertext linking strategies”. ACM 1992, pp. 221–231.*

L. Katz, “A new status index derived from sociometric analysis,” 1953, Psychometricka, vol. 18, pp. 39–43.

C.H. Hubbell, “An input–output approach to clique identification sociometry,” 1965, pp. 377–399.

Mizruchi et al., “Techniques for disaggregating centrality scores in social networks,” 1996, Sociological Methodology, pp. 26–48.

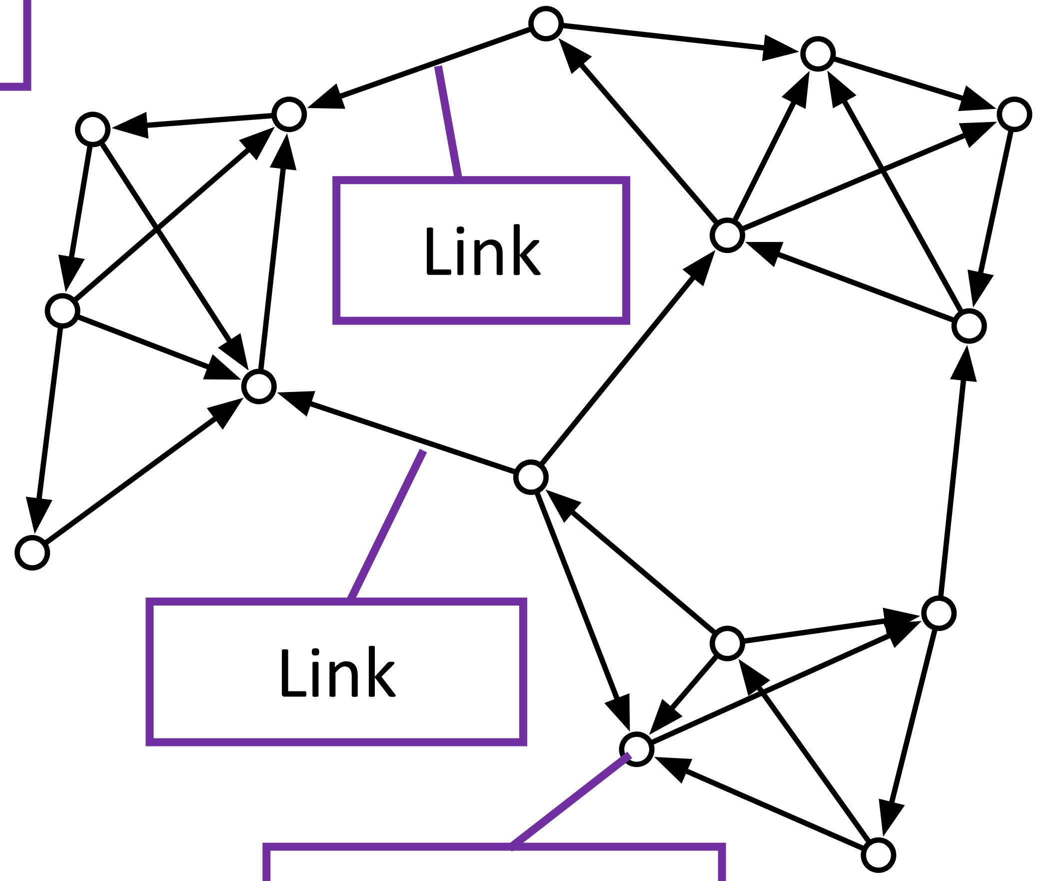
E. Garfield, “Citation analysis as a tool in journal evaluation,” 1972, Science, vol. 178, pp. 471–479.

Pinski et al., “Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics,” 1976, Inf. Proc. And Management, vol. 12, pp. 297–312.

N. Geller, “On the citation influence methodology of Pinski and Narin,” 1978, Inf. Proc. And Management, vol. 14, pp. 93–95.

P. Doreian, “Measuring the relative standing of disciplinary journals,” 1988, Inf. Proc. And Management, vol. 24, pp. 45–56.

Ranking Web Pages with PageRank



Model as a **graph**

Web page

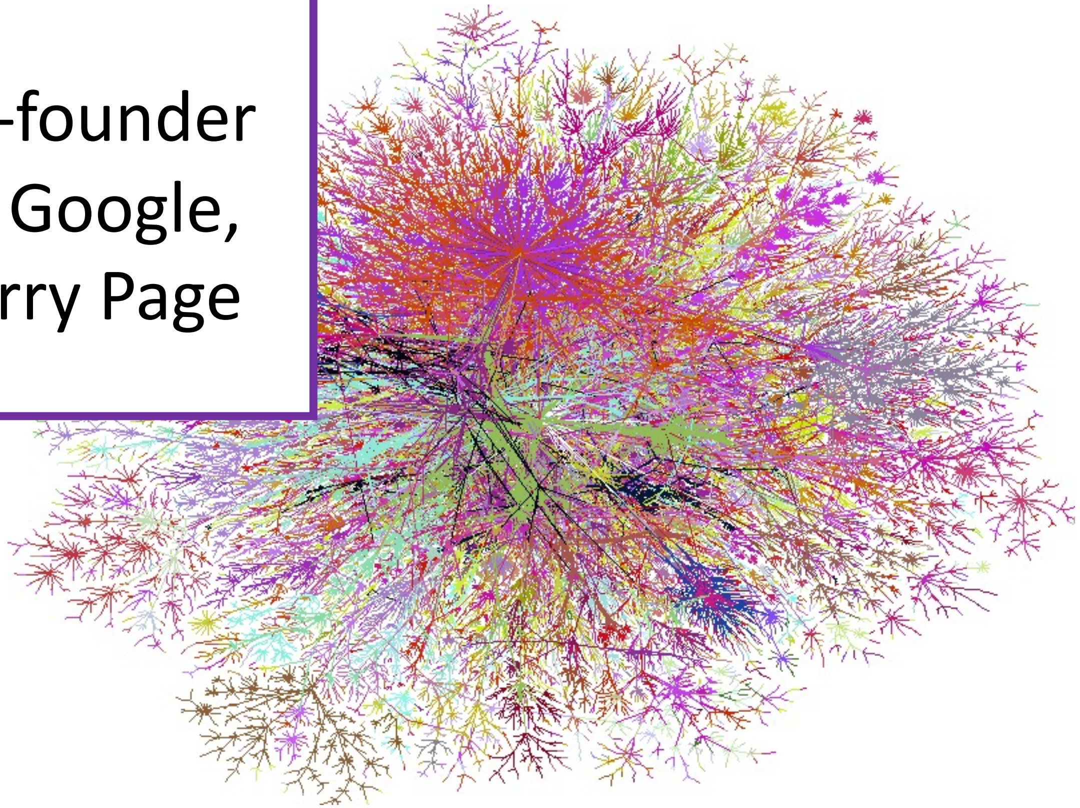
Link

Link

Web page

How PageRank came from?

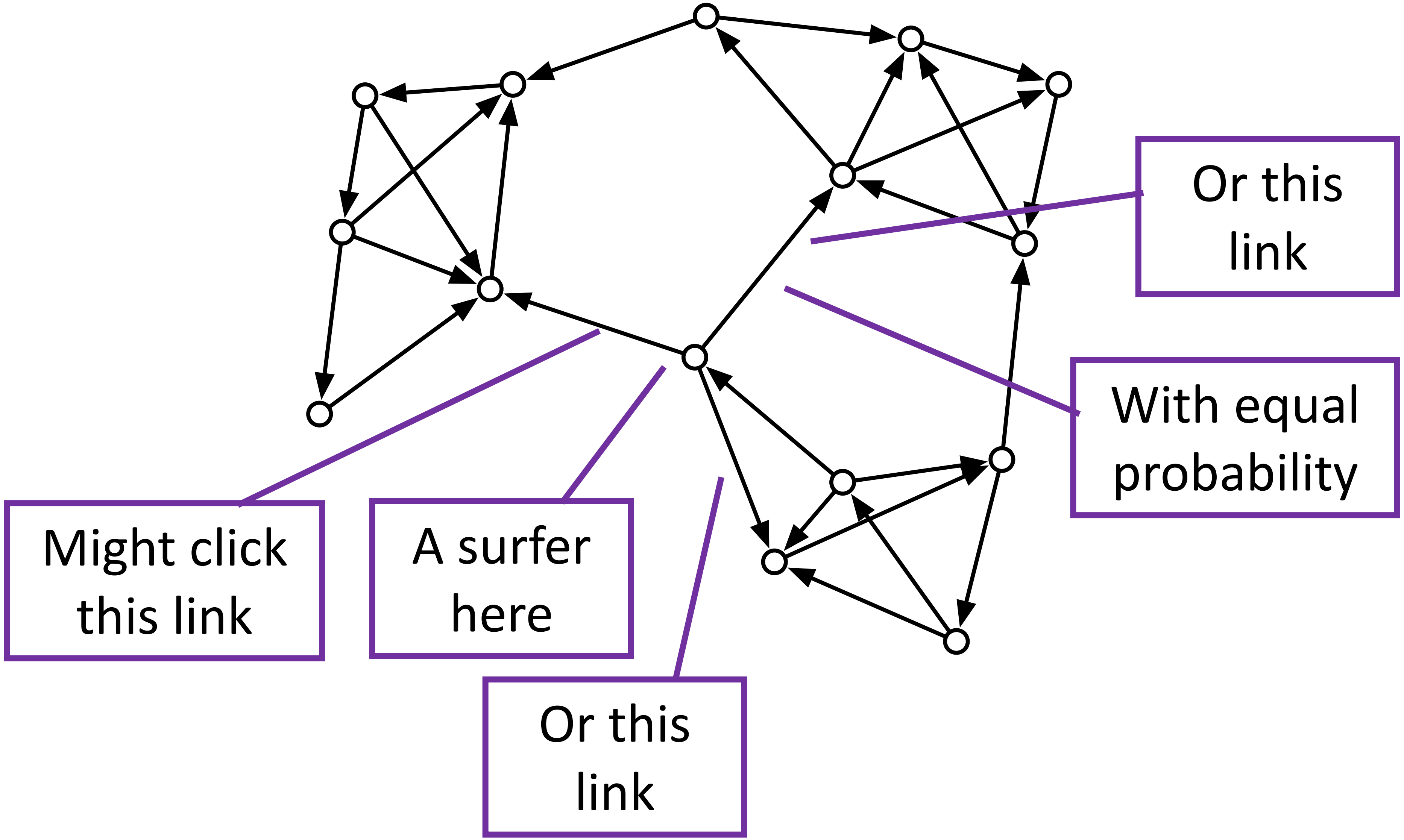
Ranking Web Pages with PageRank



Co-founder of Google, Larry Page

Web page

Surfing: Random Walk on the Web Graph



Might click this link

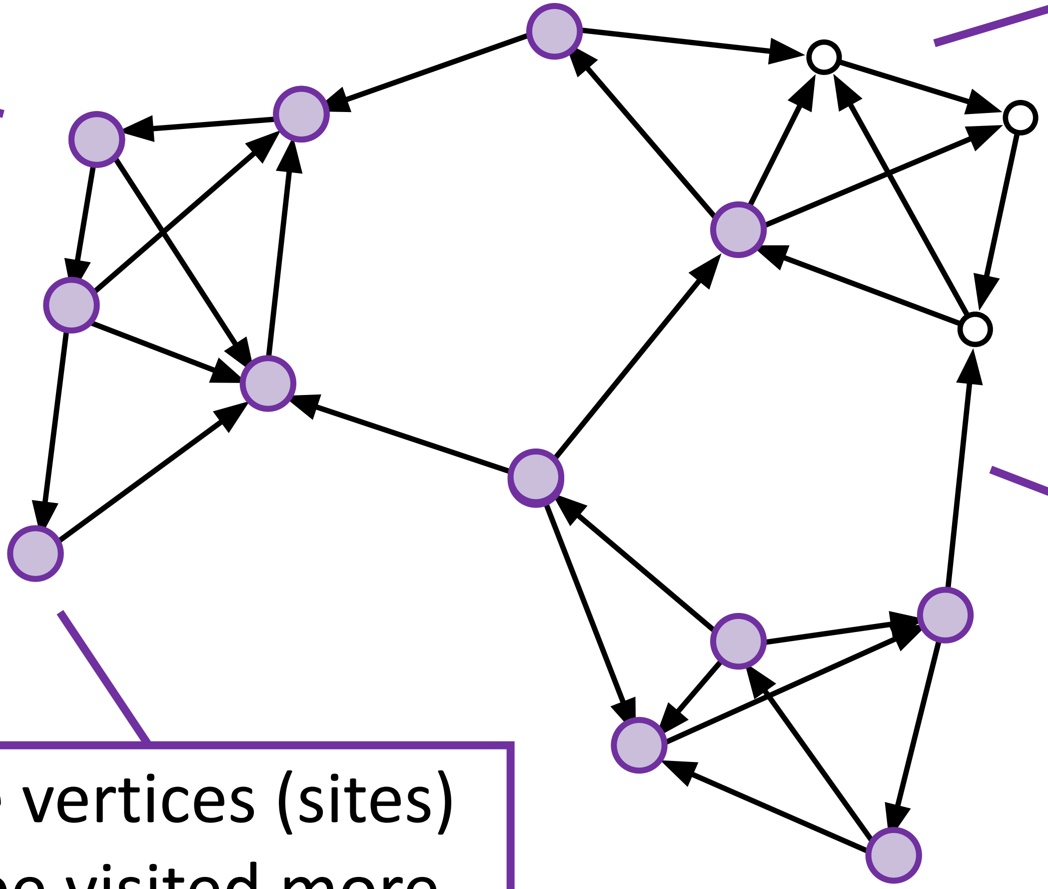
A surfer here

Or this link

With equal probability

Or this link

Surfing: Random Walk on the Web Graph



If we do this for a long time

"Important" vertex (site)

Some vertices (sites) will be visited more often than others

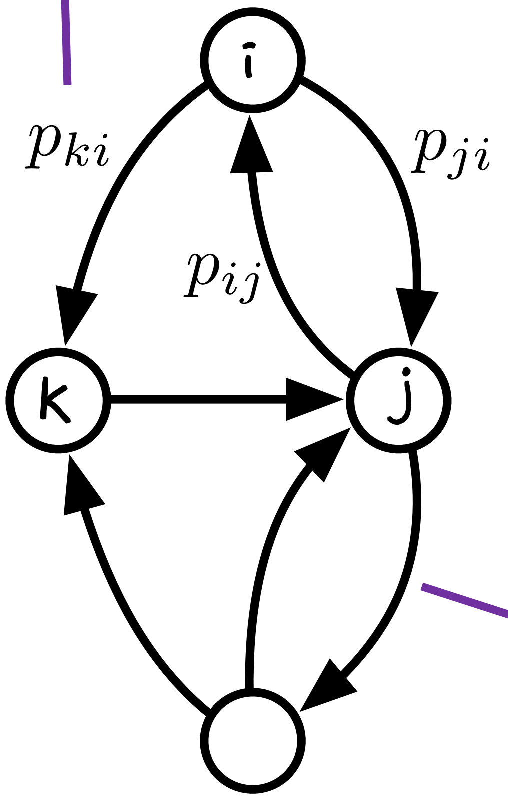
Modified random walk includes "teleportation"

PageRank: Order vertices by importance

Vector Representation

Probability that user will follow link from i to k

Probability that user will follow link from i to k



$$p_{ji} + p_{ki} = 1$$

Stochasticity

Graph of links

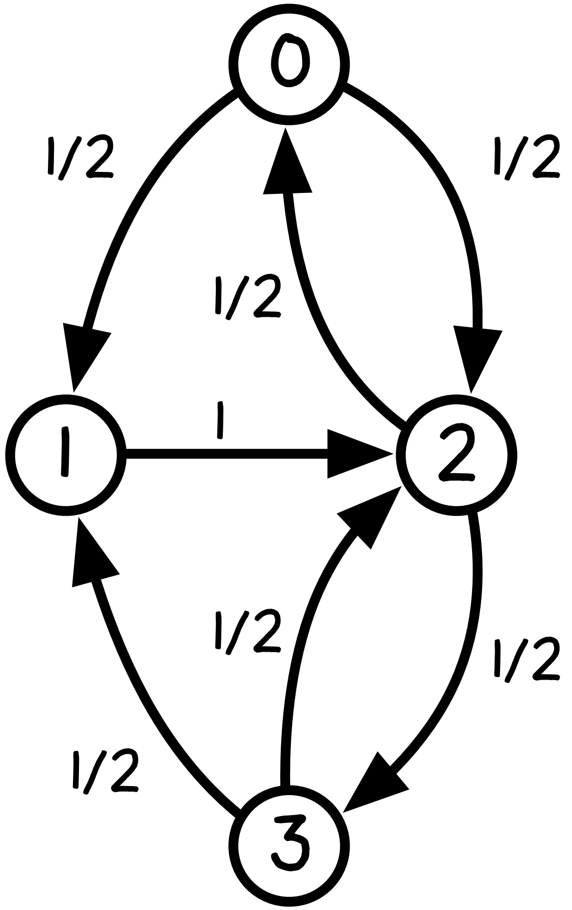
Stochastic (column) vector for node i

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ p_{ji} \\ 0 \\ \vdots \\ 0 \\ p_{ki} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Entry at row j for edge from i

Entry at row k for edge from i

Matrix Vector



Node 0

$$\begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

Node 2

$$\begin{bmatrix} \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

Node 1

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Node 3

$$\begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

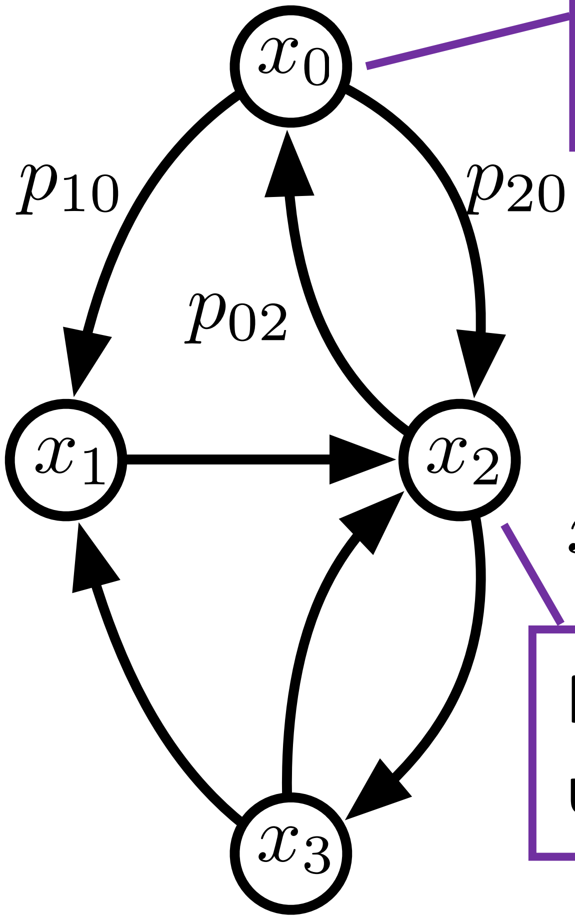
Probability that user will follow link from i to k

$$\begin{bmatrix} 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 1 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

Put vectors together into a matrix

$$\sum_i p_{ij} = 1 \quad \forall j$$

Random Walk / Markov Process



Probability user is at 0

Probability user moves from 0 to 2

Probability user is at 2

$$x_2 = p_{20}x_0 + p_{21}x_1 + p_{23}x_3$$

$$x_i = \sum_j p_{ij}x_j$$

x is an eigenvector of P

What is the eigenvalue?

$$x = Px$$

$$\sum_i p_{ij} = 1 \quad \forall j$$

$$\sum_j x_j = 1$$

Some Facts

- Exploit $\sum_i p_{ij} = 1 \ \ \forall j$ and consider left eigenvalues (which are same as right eigenvalues
- By Gershgorin, all (left) eigenvalues are in or on a circle of radius 1
- That is, spectral radius is equal to unity
- By Perron-Frobenius, there is a unique eigenvalue at the spectral radius (there is unique eigenvalue equal to unity)

- Conclusion, there is an x that satisfies $x = Px$

Computing Solution

- Let $\tilde{x} = P\tilde{x}$
- Claim

$$\lim_{k\to\infty} P^k y = \tilde{x} \quad \text{for any y}$$

So: $\tilde{x} = z$

Let

$$z = \lim_{k\to\infty} P^k y$$

Then

$$z = \lim_{k\to\infty} P^k y$$

$$= \lim_{k\to\infty} PP^k y$$

$$= P \lim_{k\to\infty} P^k y$$

$$= Pz \;\Rightarrow\; z = Pz$$

But $\tilde{x}$ is unique

Computing Solution

Matrix-matrix product (k of them)

Matrix-vector product (k of them)

$$\lim_{k \to \infty} P^k y = \tilde{x} \quad \text{for any y}$$

$$(P^k)x = P(P(P \dots (Px)))$$

Expensive!

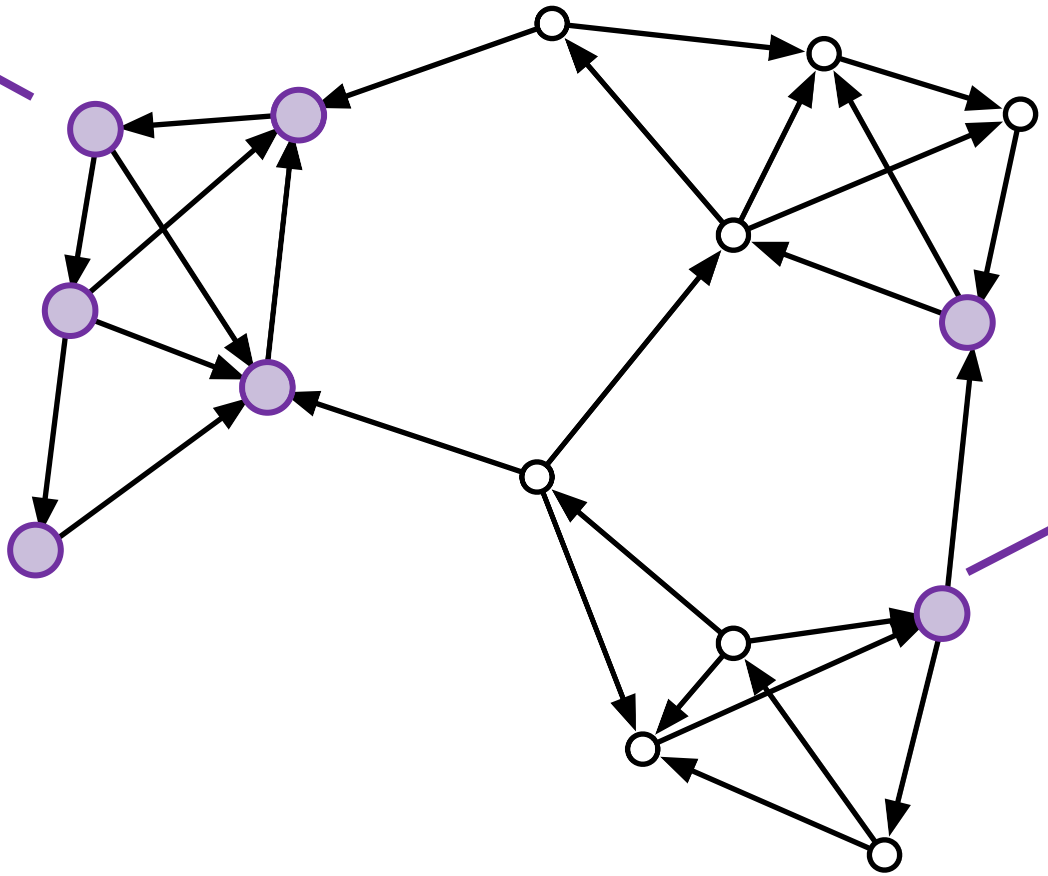
Much cheaper!

```
Vector x(N);
randomize(x);
x = (1.0 / one_norm(x)) * x;

for (size_t i = 0; i < max_iters; ++i) {
  Vector y = P * x;
  if (two_norm(x - y) < tol) {
    return y;
  }
  x = y;
}
```

Known as Power Method

Teleportation

Once we get into this cycle we can't get out



PageRank includes "teleportation"

Teleportation

Include teleportation computationally

Small probability that user might go from a site to any other site

$$Q = \frac{\alpha}{N_p}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix} + (1-\alpha)P$$

Scale to maintain Markov chain properties

Sum of all elements in column is equal to unity

Simplifying Teleportation

$$\frac{1}{N_p}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix} x = \frac{1}{N_p}\begin{bmatrix} |x|_1 \\ |x|_1 \\ \vdots \\ |x|_1 \end{bmatrix} = \frac{1}{N_p}\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$x \leftarrow (1-\alpha)Px + \frac{\alpha}{N}$$

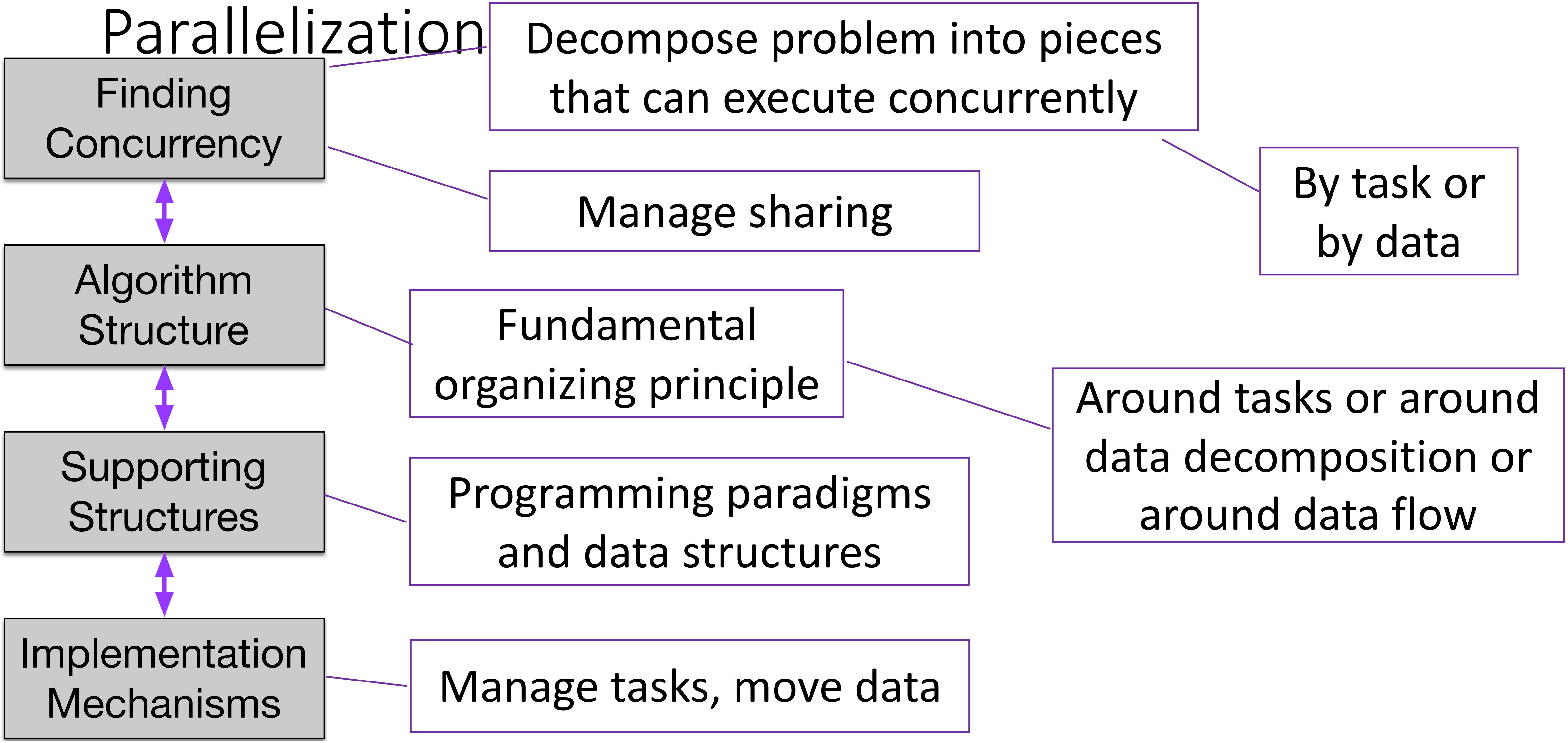
Small bias

Algorithm with Teleportation

```
Vector x(N);
randomize(x);
x = (1.0 / one_norm(x)) * x;

for (size_t i = 0; i < max_iters; ++i) {
  Vector y = (1.0 - alpha) * P * x + alpha / x.num_rows();
  if (two_norm(x - y) < tol) {
    return y;
  }
  x = y;
}
```

Teleportation bias

Parallelization



Finding Concurrency

Algorithm Structure

Supporting Structures

Implementation Mechanisms

Decompose problem into pieces that can execute concurrently

By task or by data

Manage sharing

Fundamental organizing principle

Around tasks or around data decomposition or around data flow

Programming paradigms and data structures

Manage tasks, move data

Timothy Mattson, Beverly Sanders, and Berna Massingill. 2004. *Patterns for Parallel Programming*(First ed.). Addison-Wesley Professional.

Thank you!

Creative Commons BY-NC-SA 4.0 License

© Andrew Lumsdaine, 2017-2022
Except where otherwise noted, this work is licensed under

https://creativecommons.org/licenses/by-nc-sa/4.0/