CME 213

Spring 2017

Eric Darve

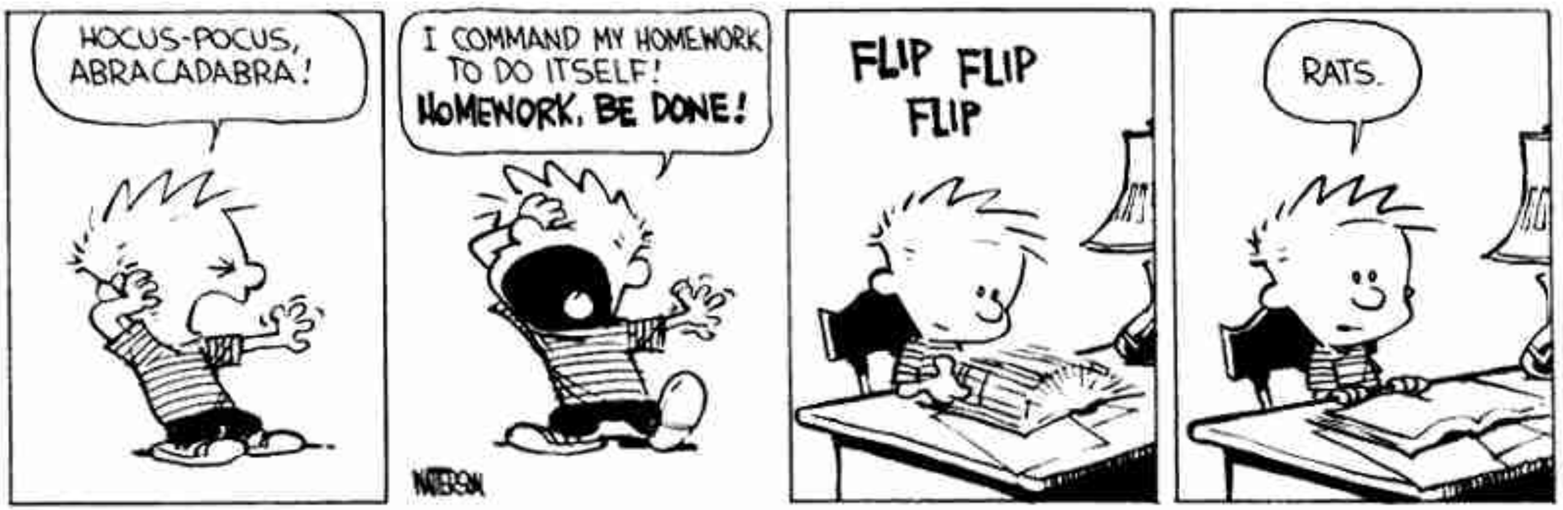
Stanford University

ANNOUNCEMENT: PROF. JINCHAO XU'S 1-UNIT CLASS

- CME 335A Optimal Iterative Methods for Linear and Nonlinear Problems
- 1-unit course meeting on 4/10, 4/12, 4/14, 4/17 and 4/19 from 3:30-5:20pm in GESB150.
- Instructor: Prof. Jinchao Xu, Verne M. Willaman Professor of Mathematics, Penn State University
- Prof. Xu is best known for an algorithm that is now one of the two most fundamental multigrid approaches for solving large-scale PDEs – the Bramble-Pasciak-Xu preconditioner – and one of the most efficient methods for solving Maxwell's equations – the Hiptmair-Xu preconditioner.

HOMEWORK INSTRUCTIONS

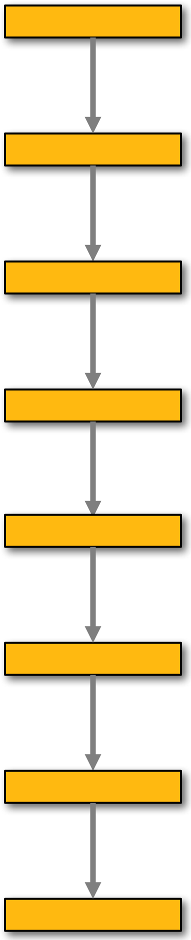
- Download homework handout and skeleton code from canvas.
- Copy all the files to `corn`
- Submit using the provided script
- Turn in computer code + PDF with text answers.
- Deadline is: Wednesday April 12th, 11pm
- There is a 24 hour grace period.


HOCUS-POCUS, ABRACADABRA!
I COMMAND MY HOMEWORK TO DO ITSELF! HOMEWORK, BE DONE!
WATTERSON
FLIP FLIP FLIP
RATS.

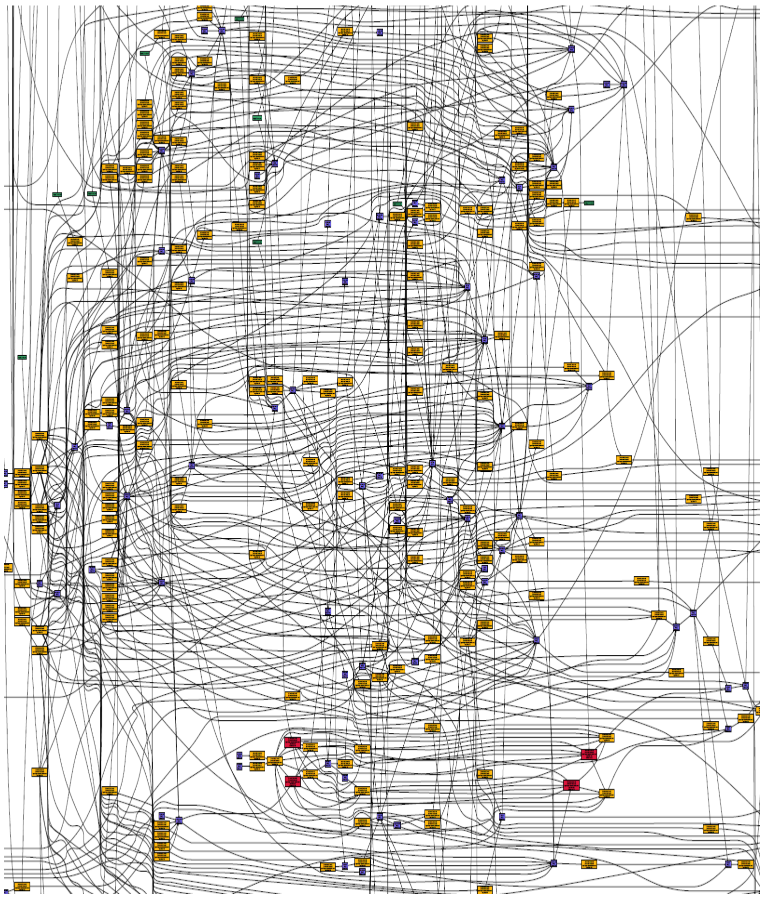
Example of Parallel Computation

WHY WE NEED TO WRITE PARALLEL PROGRAMS

- Most programs you have written so far are (probably) sequential.
- Unfortunately parallel programs often look very different…



Sequential program



Parallel program

- An efficient parallel implementation of a serial program may not be obtained by simply parallelizing each step.
- Rather, the best parallelization may be obtained by stepping back and devising an entirely new algorithm.

LET’S CALCULATE THE SUM OF N NUMBERS

```
sum = 0;
for (i = 0; i < n; i++) {
   x = ComputeNextValue(…);
   sum += x;
}
```

OUR FIRST PARALLEL PROGRAM

- Assume we have p cores that can compute and exchange data.
- Then we could accelerate the previous calculation by splitting the work among all these cores.

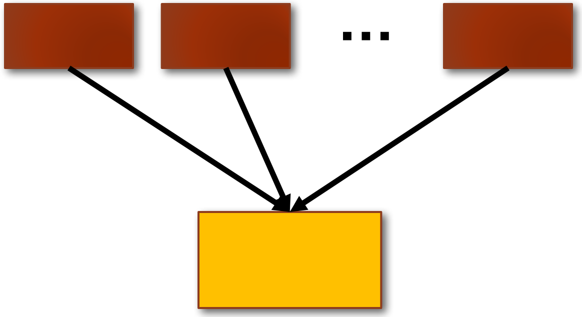
```
my_sum = 0;
my_first_i = … ;
my_last_i = … ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
    my_x = ComputeNextValue(…);
    my_sum += my_x;
}
```

BUT IT'S NOT THAT SIMPLE

- Each core has computed a partial sum.
- All these partial sums need to summed up together.
- The simplest approach is to have one "master" core do all the work:

In pseudo-code:

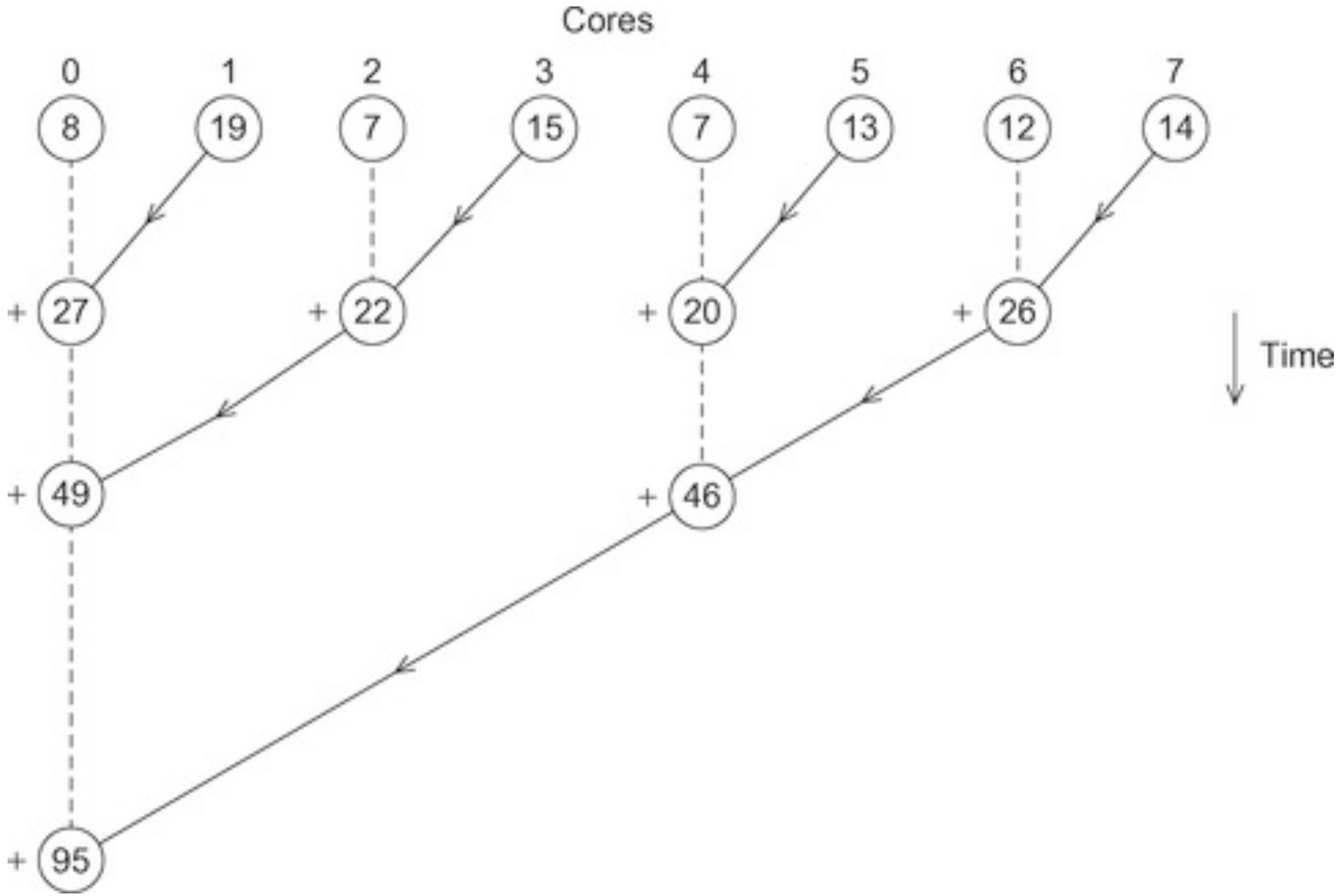
```
if (I am the master core) {
   sum = my_x;
   for each core other than myself {
      receive value from core;
      sum += value;
   }
} else {
   send my_x to the master;
}
```



That may not be enough

- If we have many cores, this final sum may in fact take a lot of time.

- How would you design a better implementation?


Cores
0
1
2
3
4
5
6
7
8
19
7
15
7
13
12
14
+ 27
+ 22
+ 20
+ 26
Time
+ 49
+ 46
+ 95

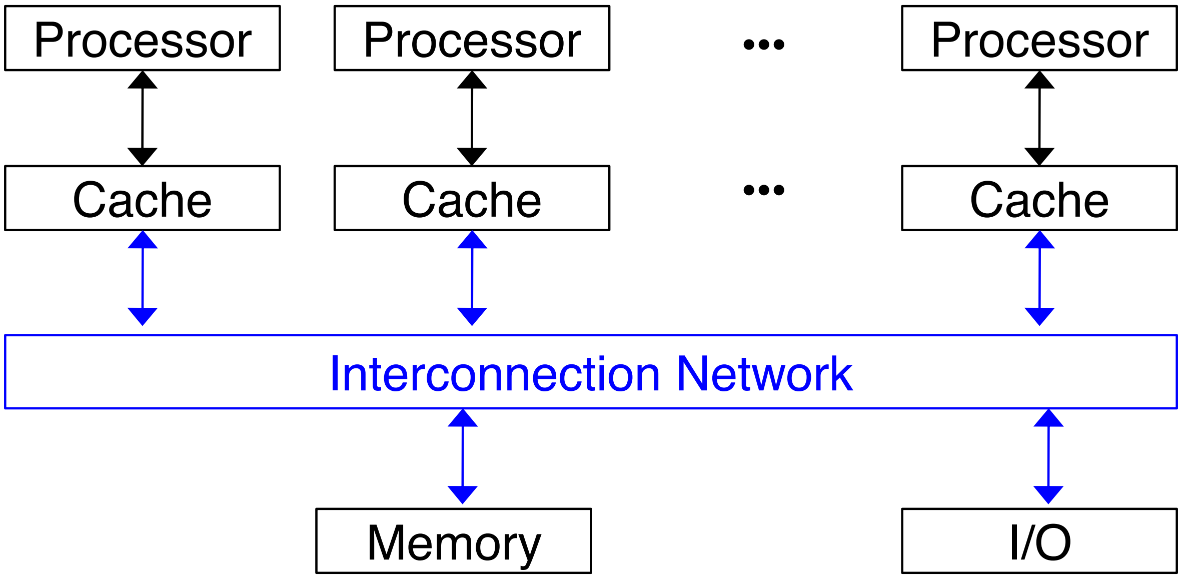
AUTOMATIC PARALLELIZATION

- This simple example illustrates the fact that it is difficult for a compiler to parallelize a program.
- Instead the programmer must often re-write his code having in mind that multiple cores will be computing in parallel.
- The purpose of this class is to teach you the most common parallel languages used in science and engineering.

Shared Memory Processor

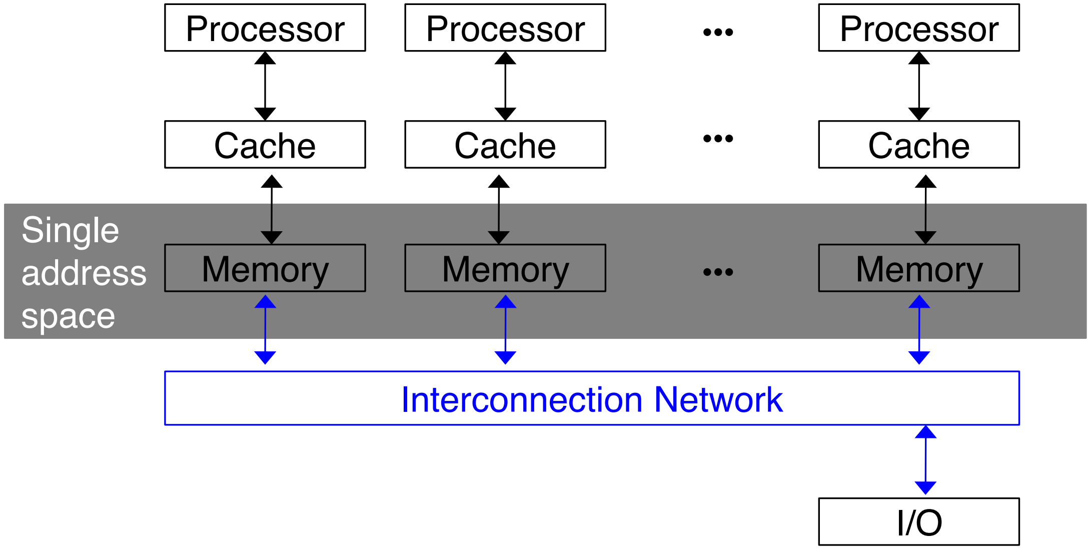
SCHEMATIC OF A MULTICORE PROCESSOR

- **Model for shared memory machines**
- **Comprised of:**
  - A number of processors or cores
  - A shared physical memory (global memory)
  - An interconnection network to connect the processors with the memory.

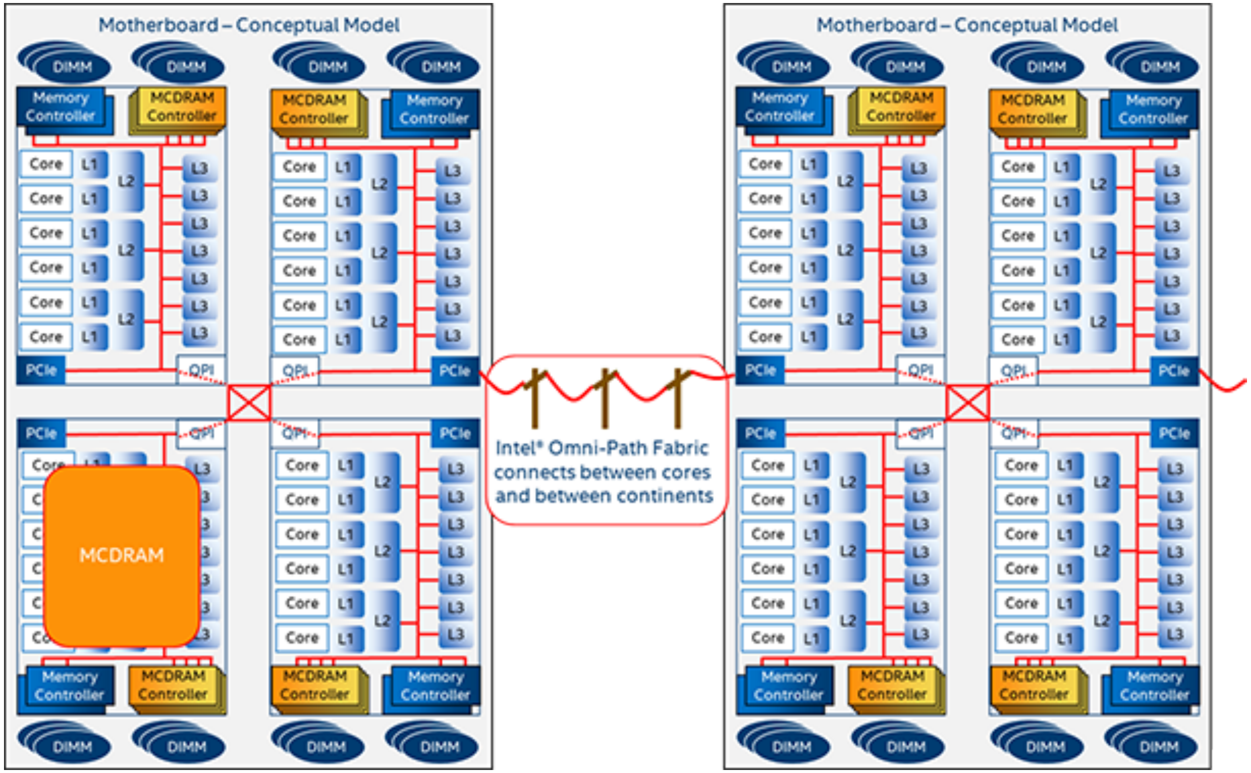


SHARED MEMORY NUMA

- In many cases, the program views the memory as a single addressable space, but in reality the memory is physically distributed.
- NUMA: non-uniform memory access.
- Faster access to memory, but special hardware required to move data between memory banks, e.g., Intel Omni-Path Fabric.



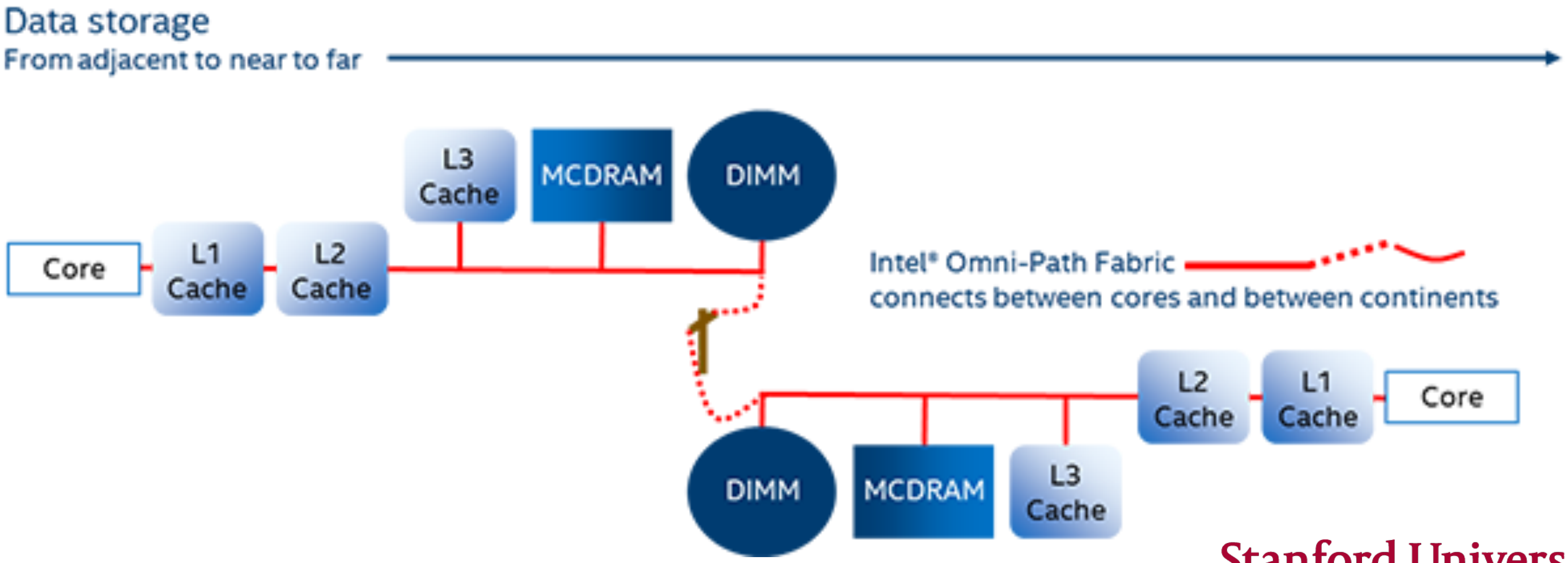
CASE STUDY: INTEL PROCESSOR


Motherboard – Conceptual Model
DIMM
Memory Controller
MCDRAM Controller
Core
L1
L2
L3
PCIe
QPI
MCDRAM
Intel® Omni-Path Fabric connects between cores and between continents

- MCDRAM: proprietary, high-bandwidth memory that physically sits atop Xeon Phi processors (Knights Landing). HBM: Xeon Phi Knights Hill.
- Omni-Path connects a core to memory sitting next to other cores.

Memory is hierarchical

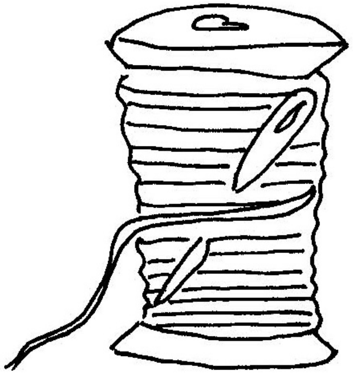
- In this class we will only briefly discuss performance for multicore processors.
- Things to keep in mind for performance:
  - memory is key to developing high-performance multicore applications
  - memory is hierarchical and complex.



SIZE, LATENCY, BANDWIDTH OF MEMORY SUBSYSTEM

Memory	Size	Latency	Bandwidth
L1 cache	32 KB	1 nanosecond	1 TB/second
L2 cache	256 KB	4 nanoseconds	1 TB/second Sometimes shared by two cores
L3 cache	8 MB or more	10x slower than L2	>400 GB/second
MCDRAM		2x slower than L3	400 GB/second
Main memory on DDR DIMMs	4 GB-1 TB	Similar to MCDRAM	100 GB/second
Main memory on Intel Omni-Path Fabric	Limited only by cost	Depends on distance	Depends on distance and hardware
I/O devices on memory bus	6 TB	100x-1000x slower than memory	25 GB/second
I/O devices on PCIe bus	Limited only by cost	From less than milliseconds to minutes	GB-TB/hour Depends on distance and hardware

PROCESSES AND THREADS

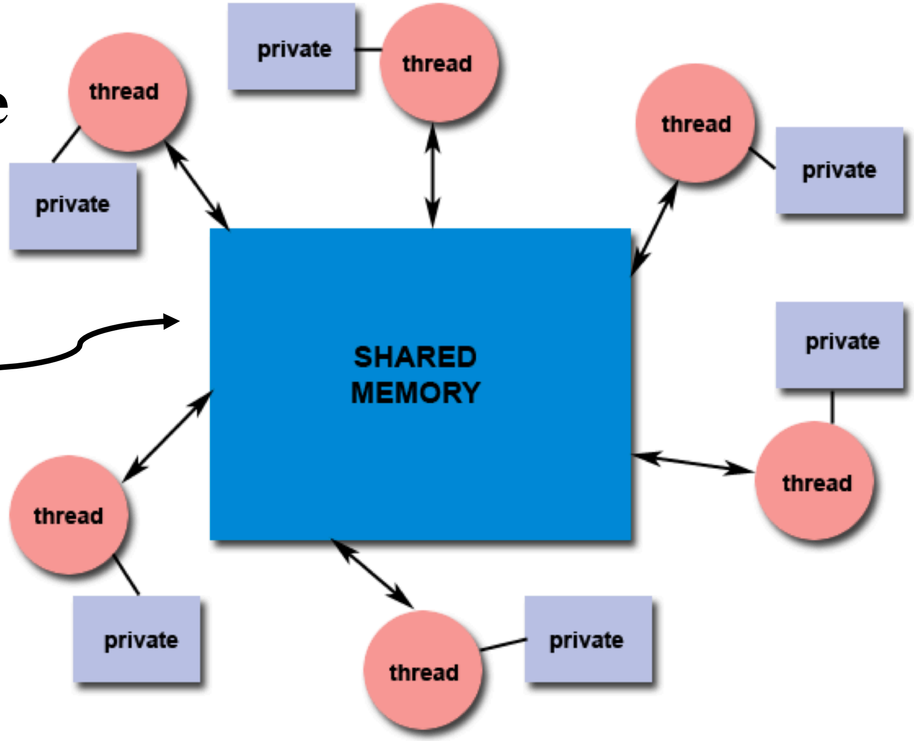


Definition:

- **Process:**
  - Program in execution.
  - Comprises: the executable program along with all information that is necessary for the execution of the program.
- **Thread: an extension of the process model. Can be viewed as a "lightweight" process.**
- **In this model, each process may consist of multiple independent control flows that are called threads.**
- **A thread may be described as a "procedure" that runs independently from the main program.**
- **Imagine a program that contains a number of procedures. Then imagine these procedures being able to be scheduled to run simultaneously and/or independently by the operating system. This describes a "multi-threaded" program.**

Shared address space

- All the threads of one process share the address space of the process, i.e., they have a common address space.
- When a thread stores a value in the shared address space, another thread of the same process can access this value.

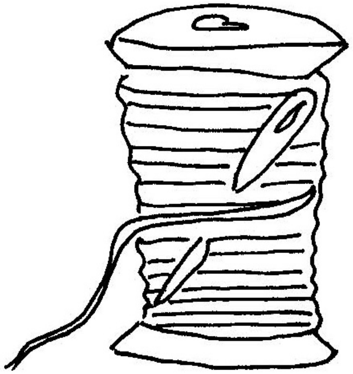


Private storage

Data exchange

PTHREADS

THREADS ARE EVERYWHERE



- C threads: Pthreads
- C++ threads (11):

```
std::thread
```

- Java threads:

```
Thread thread = new Thread();
```

- Python threads:

```
t = threading.Thread(target=worker)
```

- Cilk:

```
x = spawn fib (n-1);
```

- Julia:

```
r = remotecall(rand, 2, 2, 2)
```

- OpenMP

PTHREADS

- This is the most "low-level" approach for programming in parallel.
- Pthreads: POSIX threads. This is a standard to implement threads on UNIX systems. It is based on the C programming language.
- Pthreads will serve as an introduction to the most important concepts in multicore programming.
- The other approach we will cover is OpenMP.
- OpenMP is the standard for multicore programming in scientific programs.
- Pthreads will help you understand OpenMP.

THE BASICS

- Include the header file:

```
include <pthread.h>
```

- Compile using:

```
gcc -o hello_pthread hello_pthread.c
    -lpthread
```

- See `hello_pthread.c`

THREAD CREATION

```
int pthread_create(
   pthread_t *thread,
   const pthread_attr_t *attr,
   void *(*routine)(void*),
   void *arg)
```

- **`thread`** thread identifier
- **`routine`** function that will be executed by the thread
- **`arg`** pointer to the argument value with which the thread function **`routine()`** will be executed
- **`attr`** use **NULL** for the time being

THREAD TERMINATION

A thread terminates when:

1. Thread reaches the end of its thread function, i.e., returns.
2. Thread calls

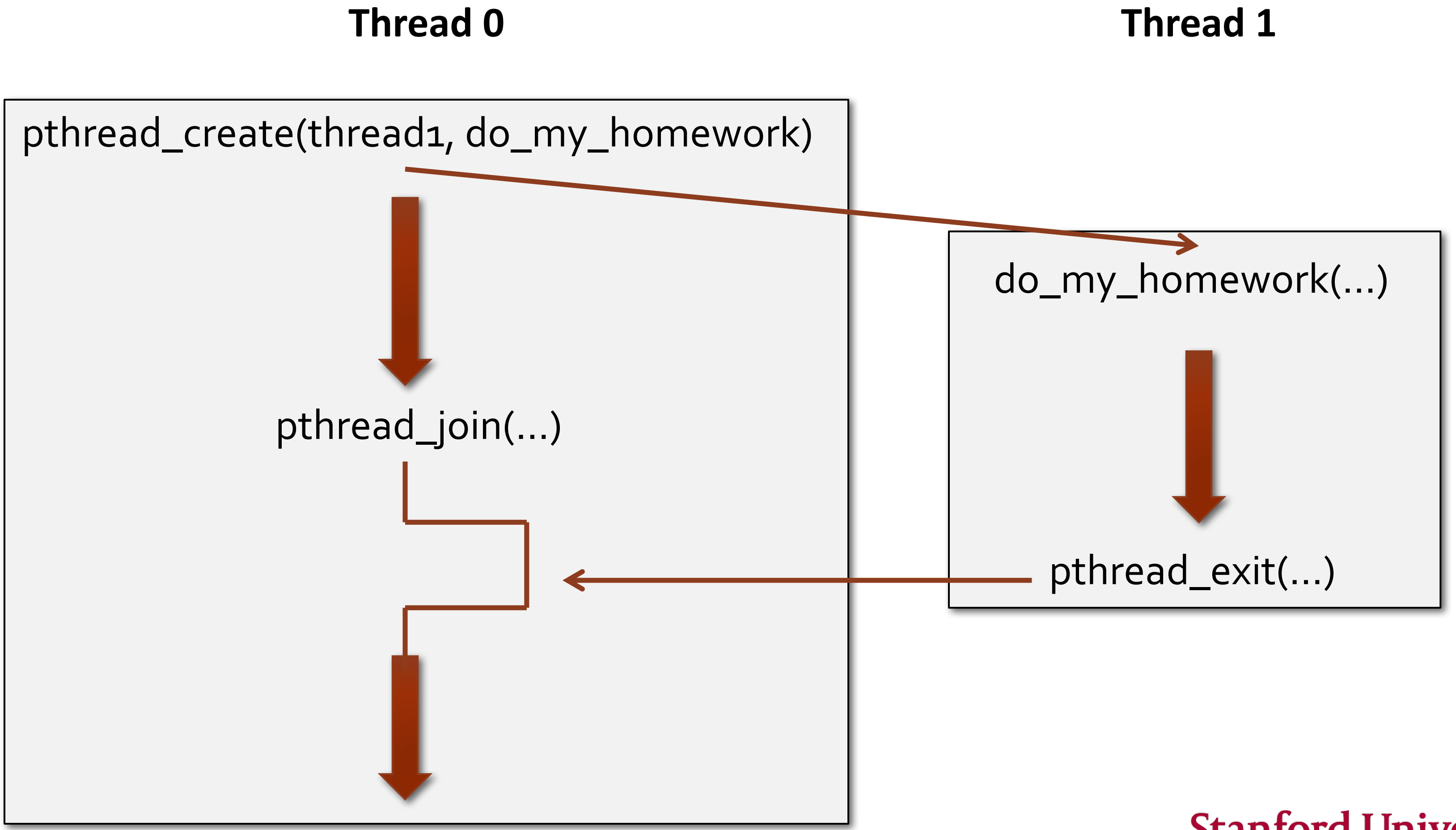
```
void pthread_exit(void *valuep)
```

Note:

- Upon termination, a thread releases its runtime stack.
- Therefore, the pointer should point to: 1) a global variable, or 2) a dynamically allocated variable.

```
int pthread_join(pthread_t thread, void **valuep)
```

- Calling thread waits for `thread` to terminate.
- `pthread_join` is used to synchronize threads.
- `valuep` memory address where the return value of thread will be stored.



See

`hello_pthread_bug_1.c`

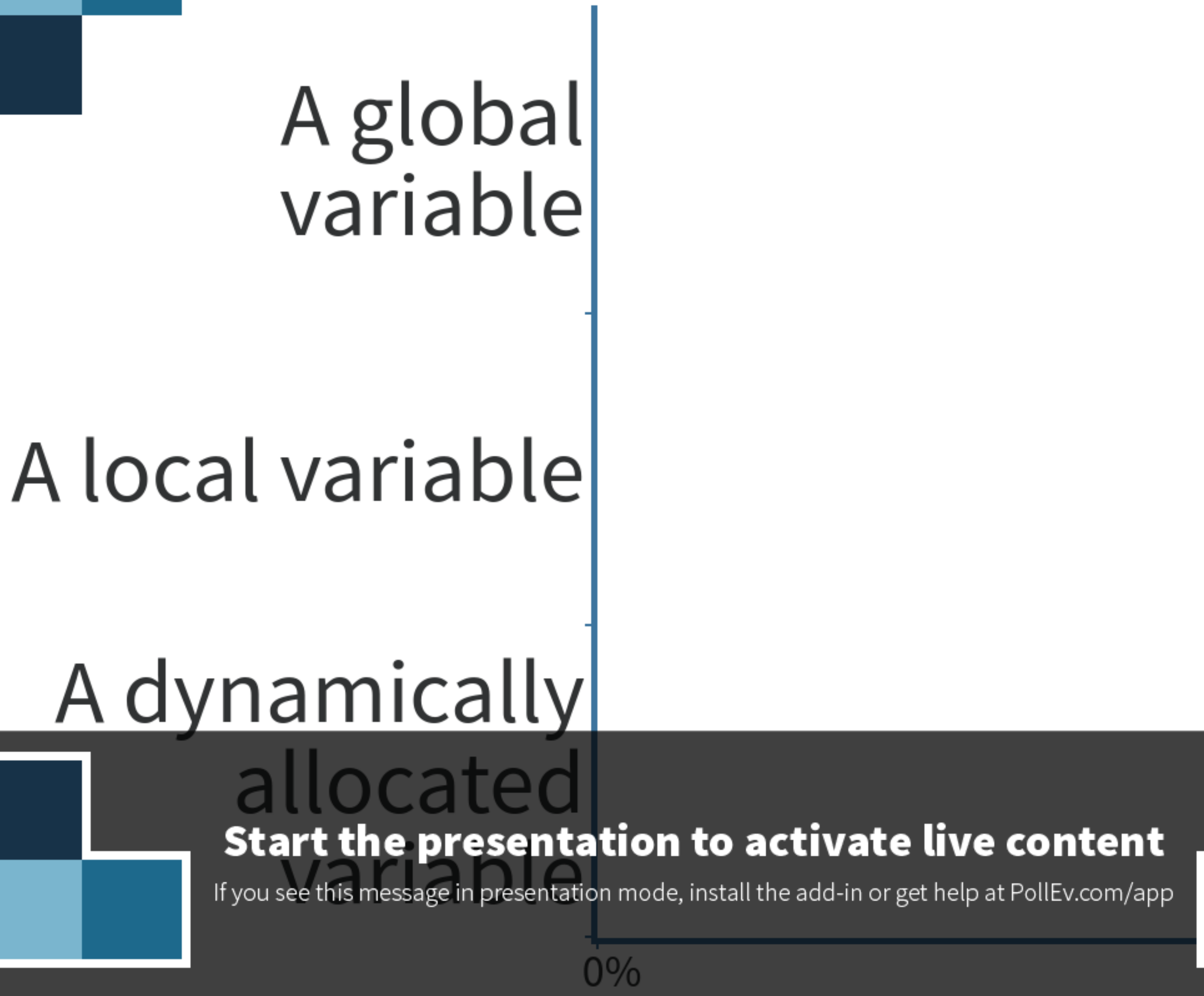
`hello_pthread_bug_2.c`

omputer, shared memory is physically in the location

Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

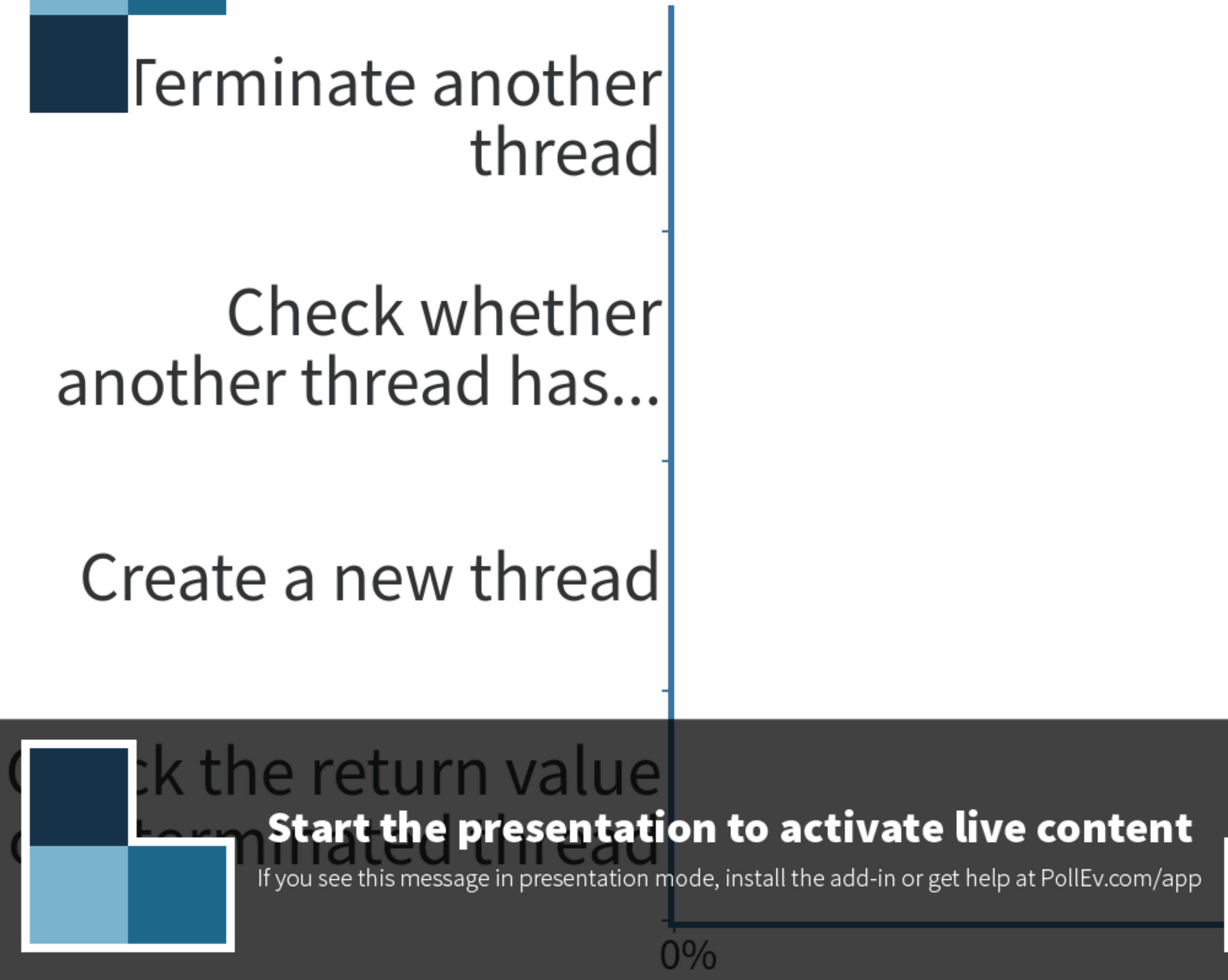
read_exit(valuep): valuep is a pointer pointin



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

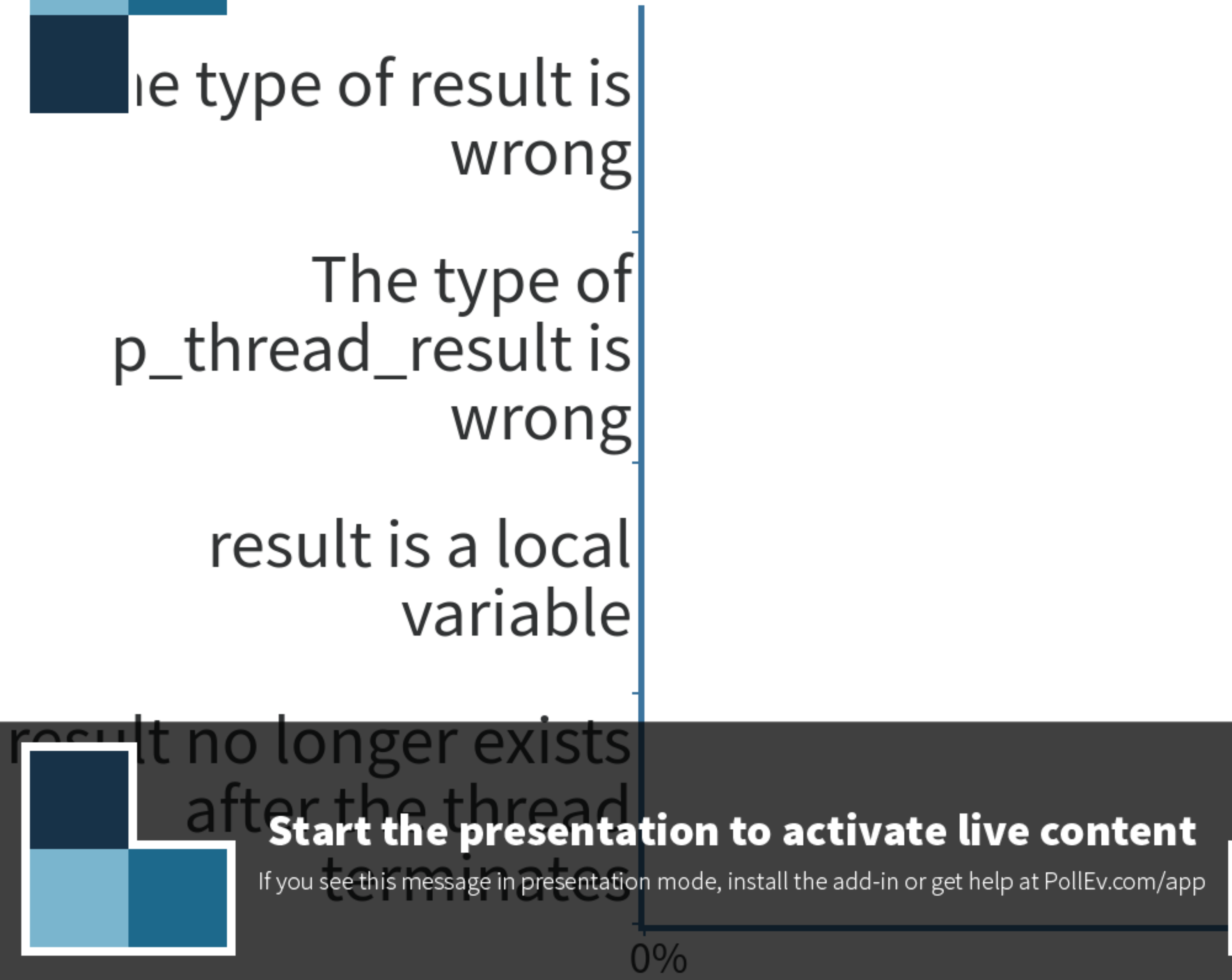
What is the role of pthread_join?



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

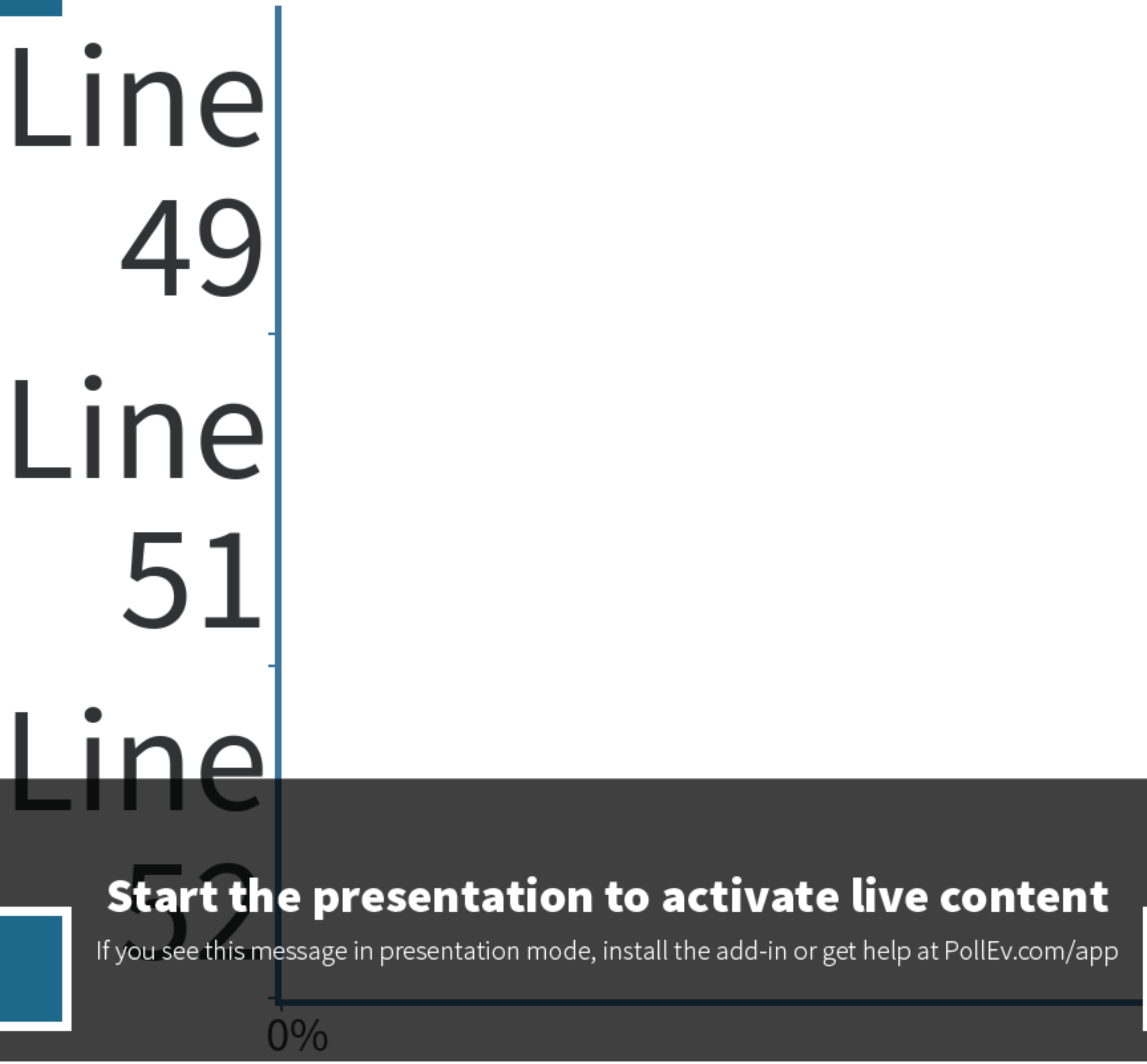
What's the error in bug_1.c?



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

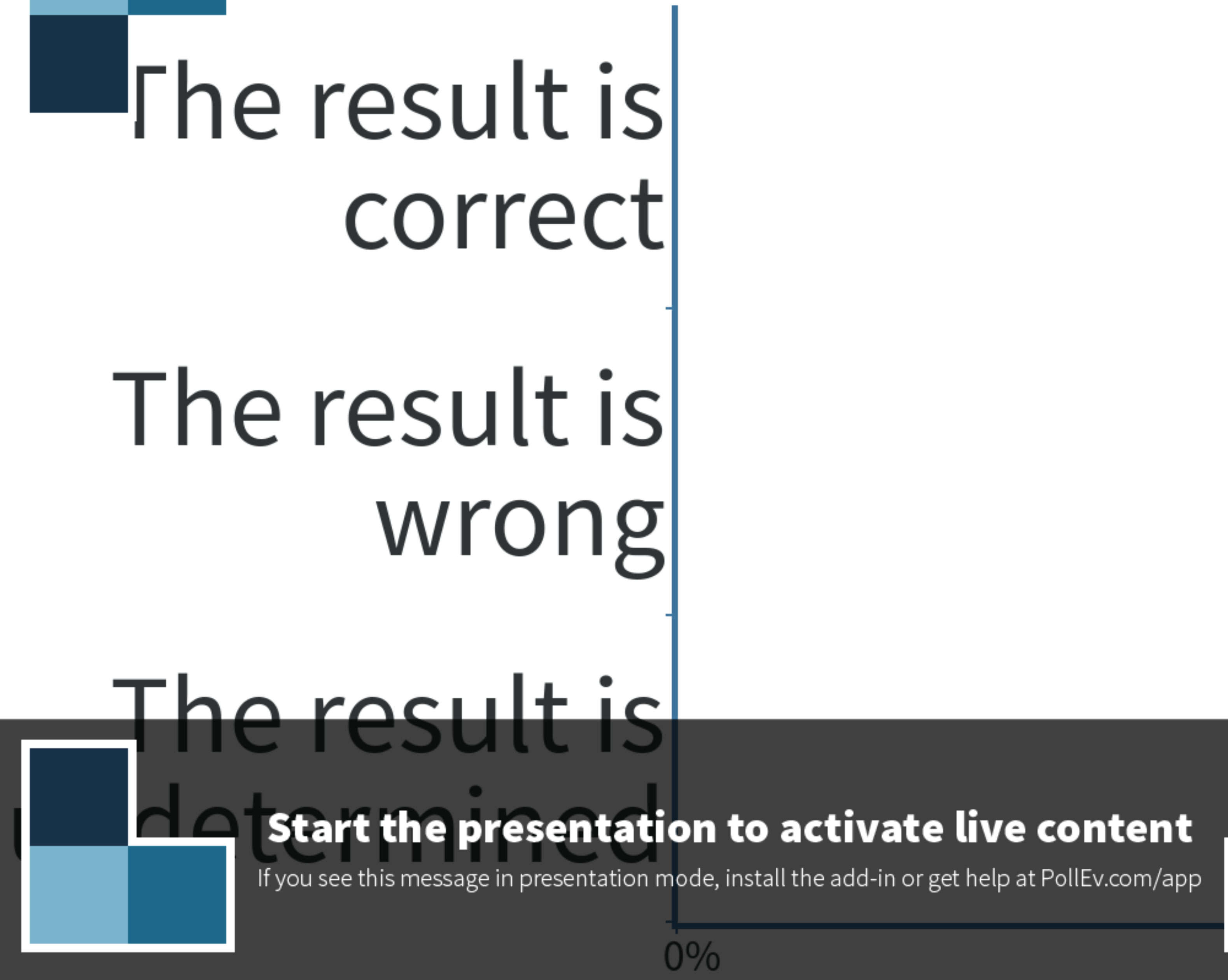
Line would you delete in order for bug_2.c to



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app

What happens if you delete line 49 in bug_2.c



Start the presentation to activate live content

If you see this message in presentation mode, install the add-in or get help at PollEv.com/app