Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Overview

- Concurrency
  - What / Why / How,
  - Grappling with Concurrency Issues.
- Operating Systems
  - Operating system architectures,
  - Memory Management (OS perspective),
  - Processes and Thread Management,
  - File Storage (disk I/O and file systems).

Practical Matters — Linux

- Course is based on Linux and the standard C program build toolset, "Build Essentials", POSIX Threads, SPIN, etc.
- Mac OS X also works pretty well.
- Windows not really supported or suitable.
- Suggest you consider running Linux in a Virtual Machine:
  - VMWare / VMWare Fusion for Mac — very good, widely used,
  - Virtual Box — free.
- Ubuntu / Debian. Don't go crazy.

Practical Matters — Labs

- 20% Labs/Practicals, 80% Examination
- Four Practicals, worth 2%, 6%, 6% and 6%.
  - First practical - simple exercise in compiling and running some concurrent code
    - Due: Monday Feb. 4th, 09:00 (via Blackboard)

Practical Matters — Software

- VMWare Academic and Microsoft Developer Licenses: https://support.scss.tcd.ie/School_Site_Licences

POSIX

- POSIX – Portable Operating System Interface
  - for variants of Unix, including Linux
  - IEEE 1003, ISO/IEC9945
- Really, considered a standard set of facilities and APIs for Unix.
  - 1988 onwards
  - Doesn't *have* to be Unix – e.g. Cygwin for Windows give it partial POSIX compliance
  - ref: http://en.wikipedia.org/wiki/POSIX

POSIX Threads

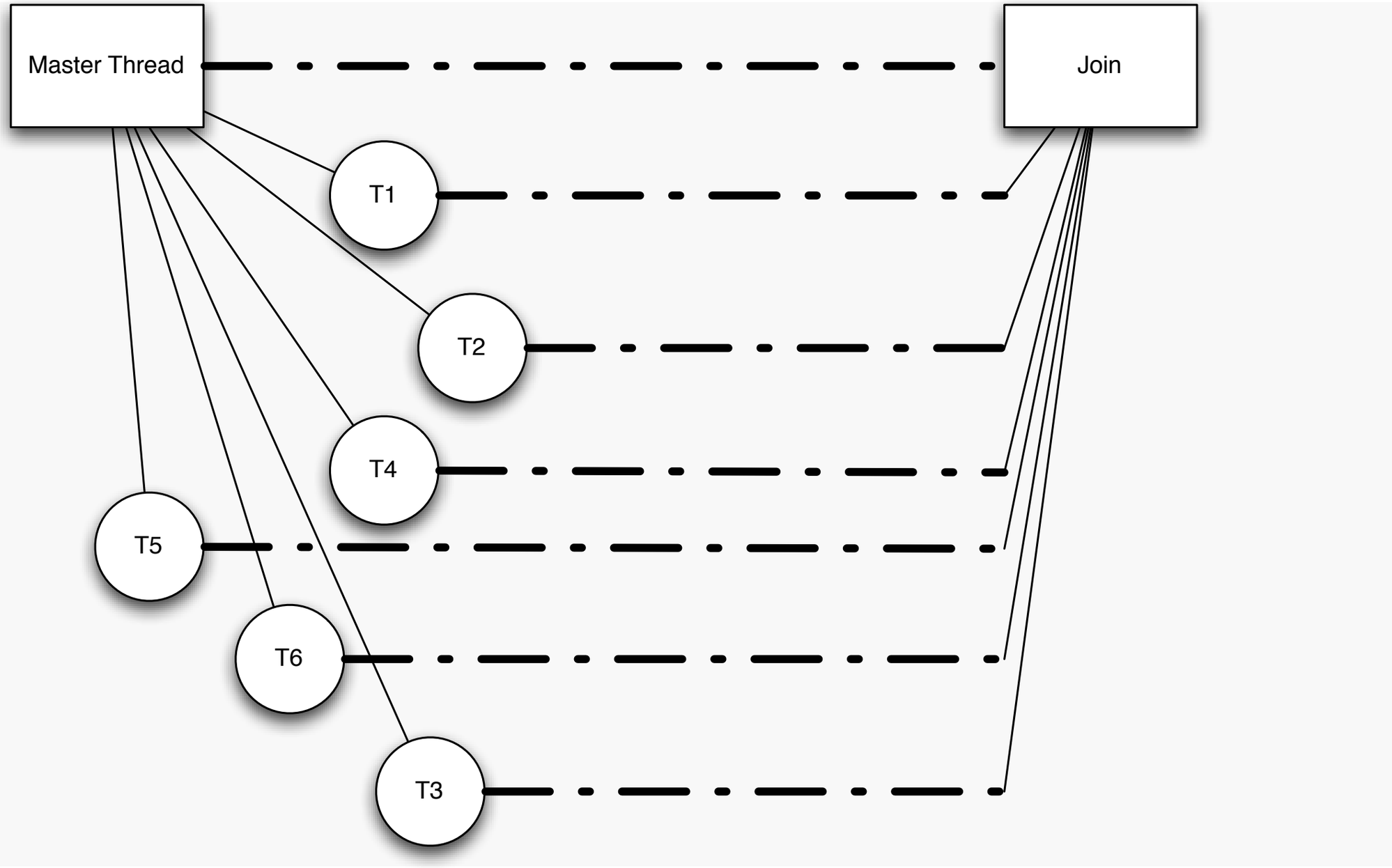
- POSIX Threads aka *'pthreads'*
  - correspond to 'Light Weight Processes' (LWPs) in older literature.
  - *pthreads* live within processes.
    - *processes* have separate memory spaces from one another
      - thus, inter-process communication & scheduling may be expensive
    - *pthreads* (within a process) share the same memory space
      - inter-thread communication & scheduling can be cheap

Classic tradeoff: speed vs. stability/ruggedness

POSIX Threads (2)

- Portable Threading Library across Unix OSes
  - All POSIX-compliant Unixes implement *pthreads*
    - Linux, Mac OS X, Solaris, FreeBSD, OpenBSD, etc.
- Also Windows:
  - E.g. Open Source: *pthreads-win32*
- BTW: Windows Threads are different!

Six Separate Threads



Creating a pthread

```
#include <pthread.h>
int pthread_create(
    pthread_t *thread,          // the returned thread id
    const pthread_attr_t *attr, // starting attributes
    void *(*start_routine)(void*),
                    // the function to run in the thread
    void *arg);  // parameter for function
```

Where…

- '*thread*' is the ID of the thread
- '*attr*' is the input-only attributes (NULL for standard attributes)
- '*start_routine*' (can be any name) is the function that runs when the thread is started, and which must have the signature:

```
void* *start_routine (void* arg);
```

- '*arg*' is the parameter that is sent to the start routine.
- returns a status code. '*0*' is good, '*-1*' is bad.

Wait for a thread to finish

- `int pthread_join(pthread_t thread, void **value_ptr);`
- where
- *'thread'* is the id of the thread you wish to wait on
- *'value ptr'* is where the thread's exit status will be placed on exit (`NULL` if you're not interested.)
- NB: a thread can be joined only to one other thread!

Hello World -- Creating Threads

```
int main (int argc, const char * argv[]) {
        pthread_t threads[NUM_THREADS];
        int rc,t;
        for (t=0;t<NUM_THREADS;t++) {
                printf("Creating thread %d\n",t);
                rc = pthread_create(&threads[t],NULL,
                                                PrintHello,(void *)t);
                if (rc) {
                        printf("ERROR return code from pthread_create(): %d\n",rc);
                        exit(-1);
                }
        }
```

Hello World -- Waiting for Exit

```
// wait for threads to exit
for(t=0;t<NUM_THREADS;t++) {
   pthread_join( threads[t], NULL);
}
```

Thread Code

```
void *PrintHello(void *threadid) {
    printf("\n%d: Hello World!\n", threadid);
    pthread_exit(NULL);
}
```

HelloWorld -- Complete

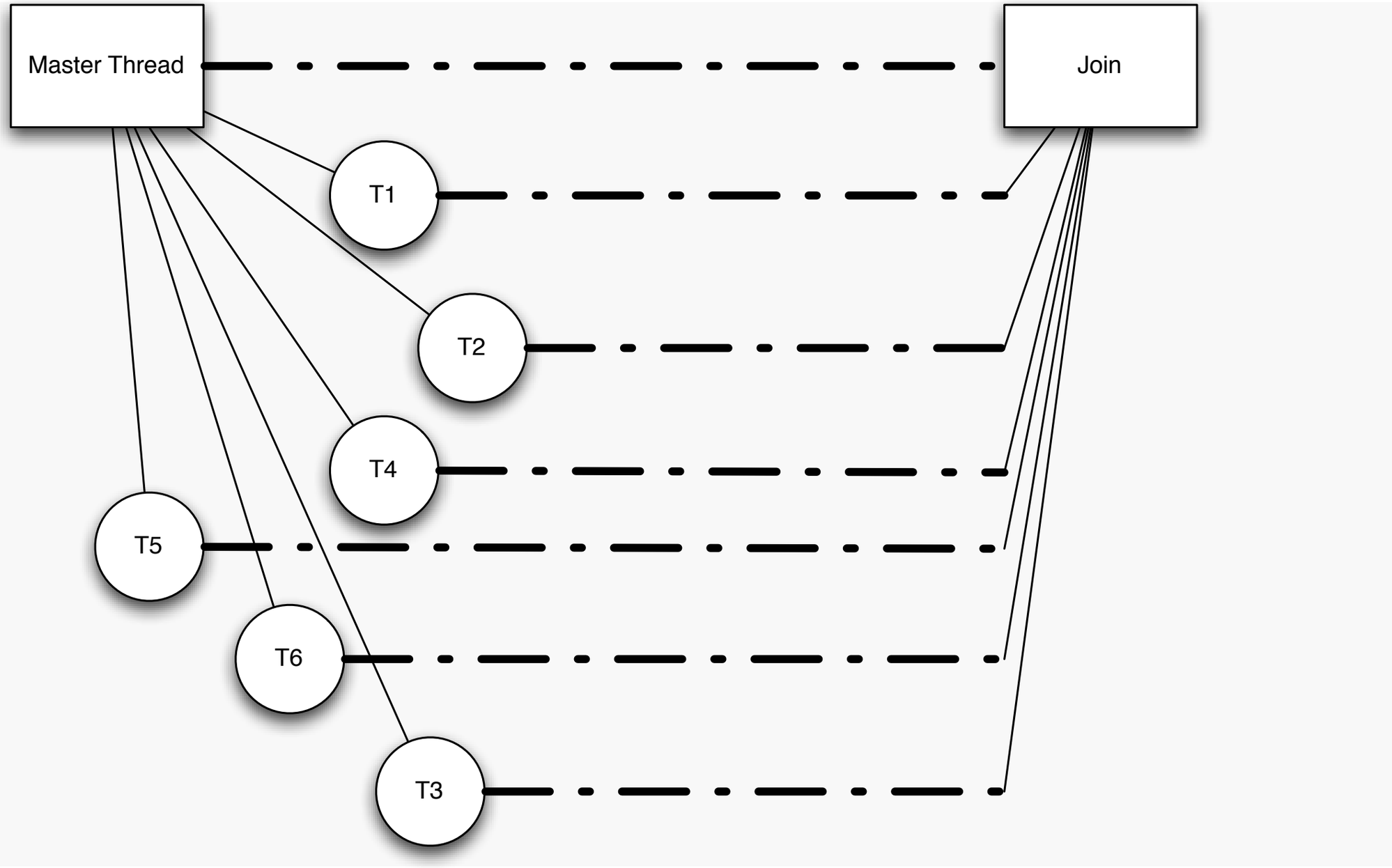
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS          6

void *PrintHello(void *threadid) {
        printf("\n%d: Hello World!\n", threadid);
        pthread_exit(NULL);
}

int main (int argc, const char * argv[]) {
        pthread_t threads[NUM_THREADS];
        int rc,t;
        for (t=0;t<NUM_THREADS;t++) {
                printf("Creating thread %d\n",t);
                rc = pthread_create(&threads[t],NULL,
                                        PrintHello,(void *)t);
                if (rc) {
                        printf("ERROR return code from pthread_create(): %d\n",rc);
                        exit(-1);
                }
        }

        // wait for threads to exit
        for(t=0;t<NUM_THREADS;t++) {
          pthread_join( threads[t], NULL);
        }
        return(0);
}
```

Turing/Stoker/Ubuntu

```
cc -o hello hello.c -pthread
```

- Include the pthread library to allow it to compile and link
  - Sometimes, -lpthread works, but -pthread is correct
- pthread library automatically included in Mac OS X build
- LIVE DEMO

What's curious about this?



What if ..?

- The threads interacted in some way?
  - e.g. each thread increments a global variable that tracks the number of threads that ran?
    - What could possibly go wrong?
  - LIVE DEMO

Runtime Behaviour

- The Runtime Behaviour of the Program is no longer under the control of the program.
  - The order in which work gets done on the machine is not exactly under the control of the program
  - It seems to be a price that's paid for parallelism, but
    - What errors can it introduce?
    - Can we prevent them / protect against them / design them out?

CS2016/3D4 plan

- POSIX Thread Programming
  - to expose you to the challenges and problems
- Understanding Concurrency
  - concepts, techniques, modelling, analysis
- Operating Systems (after Study/Review week)