COL380

Introduction to
Parallel & Distributed Programming

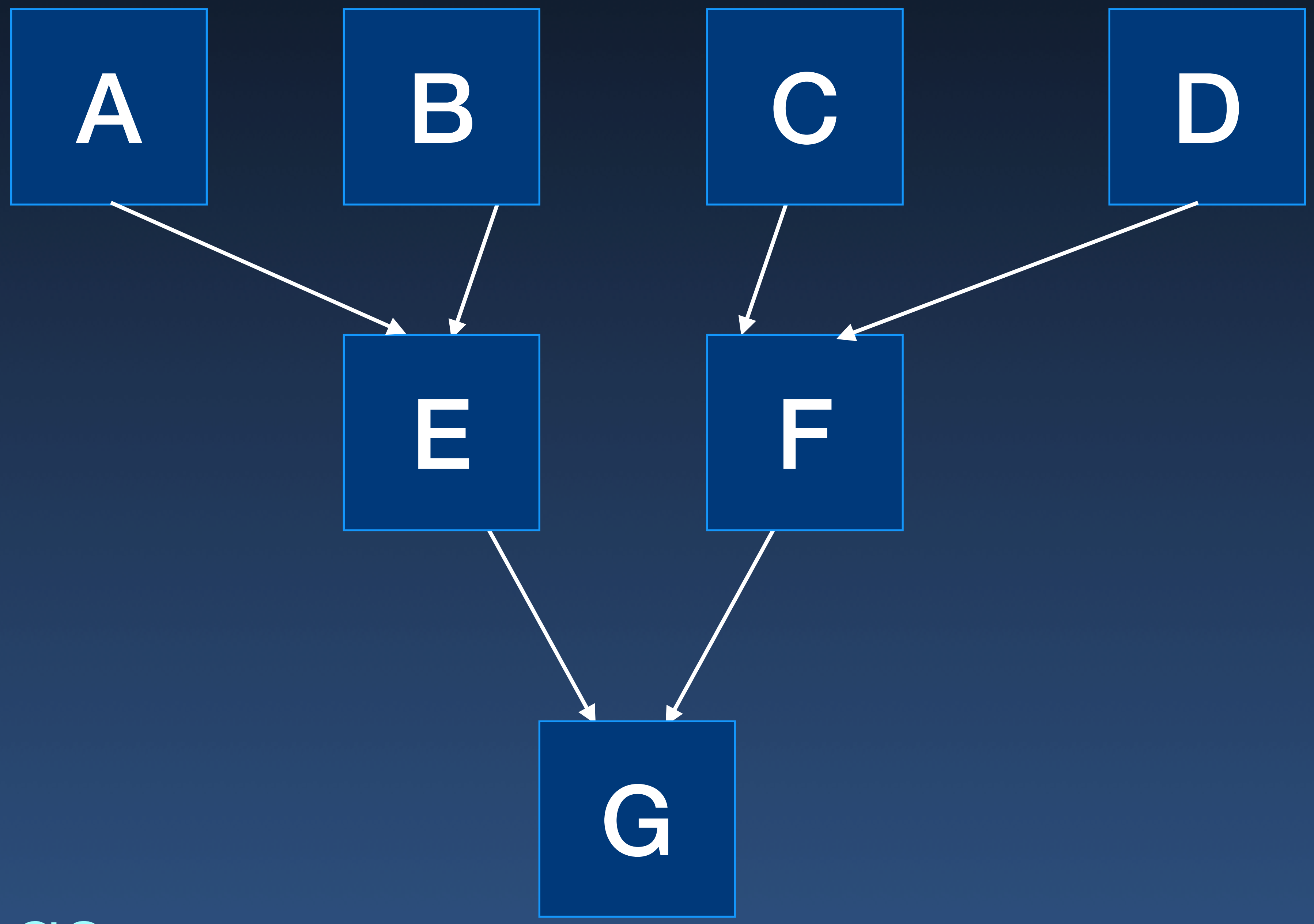
Agenda

- Task graphs and parallel programming models
- Shared memory programming
- Concurrency and race conditions
- Memory and Consistency

- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things

Parallel Programs

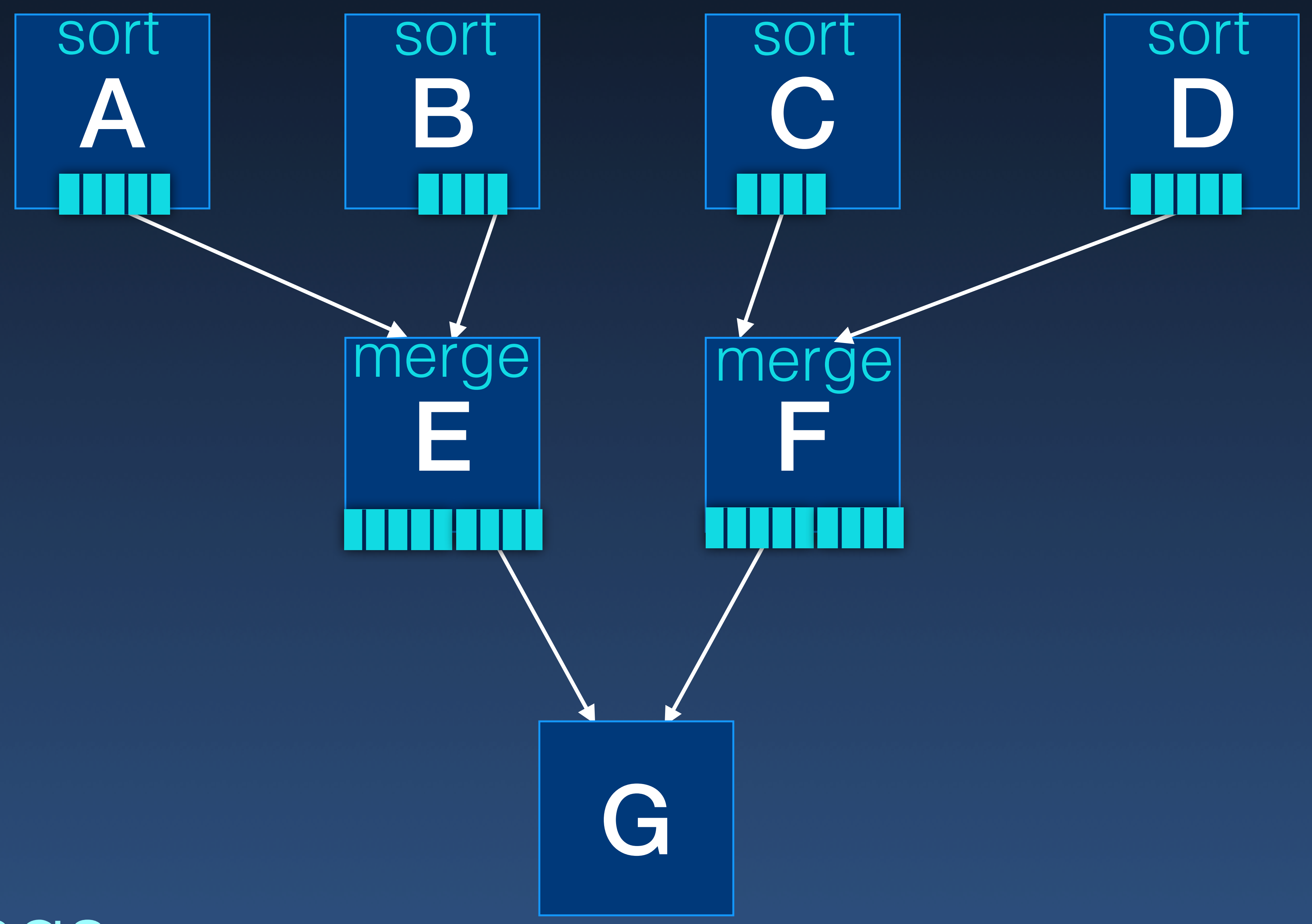
- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Program can be represented as a graph

Parallel Programs

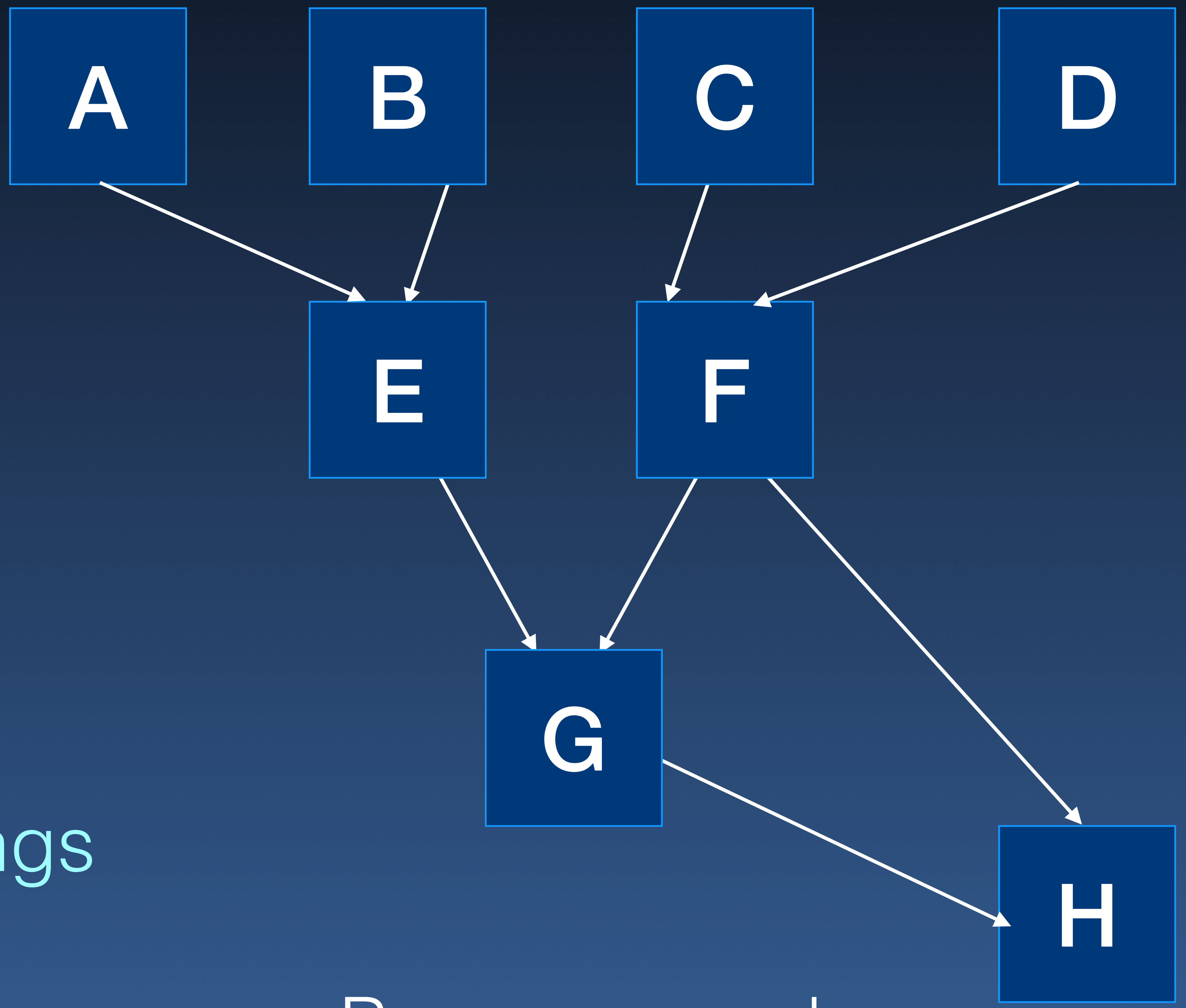
- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Program can be represented as a graph

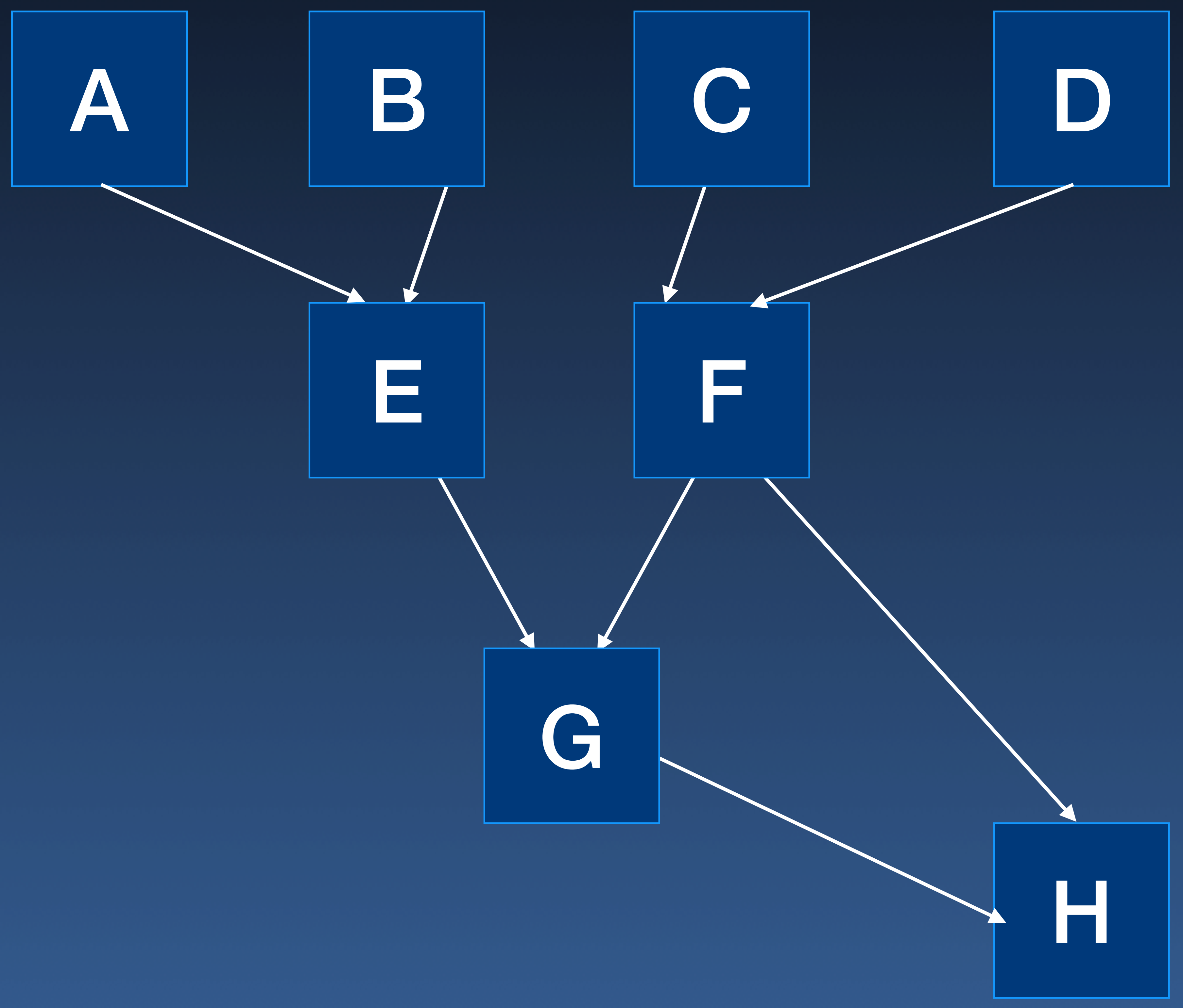
Parallel Programs

- Multiple interacting fragments
  - ➡ Shared state
  - ➡ Fragments may have private (hidden) state
- Usually Asynchronous
  - ➡ Synchronous models can simplify some things



Program can be represented as a graph

Task Graph



- Granularity
- Critical Path Length
- Maximum Concurrency
- Average Concurrency

$$= \frac{\#\text{tasks}}{\text{Critical path length}}$$

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Fork-join model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Fork-join model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

```
void processAB (int a[], int b[], int n)
{
    for(int i=0; i<n; i += 3) {
        #pragma omp task
            subtaskA(a+i);
        #pragma omp task
            subtaskB(b+i);
    }
}
......
#pragma omp task
    processAB(a, b, n);
```

Programming Models

- Shared Memory model
- Distributed Memory/Message passing model
- Task-graph based model
- Fork-join model
- Work-queue model
- Stream processing model
- Map-reduce model
- Client-server model

```
void processAB (int a[], int b[], int n)
{
    for(int i=0; i<n; i += 3) {
        #pragma omp task  depend(out: x)
            subtaskA(a+i);
        #pragma omp task  depend(in: x)
            subtaskB(b+i);
    }
}
......

#pragma omp task
    processAB(a, b, n);
```

Subodh Kumar

Programming Models: A Broad Classification

- Shared Memory
  - ➡ Tasks share a common address space they access asynchronously
    - ▸ Can Synchronize to enforce certain order
  - ➡ Implicit interaction through memory
  - ✳ Data may be cached on the processor executing the task
- Message Passing
  - ➡ Tasks have their own address spaces
  - ➡ Explicit Inter-process interaction (in code): <send> and <receive>

Share Occasionally

```
void loop(int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}             (Rd a; Add1; Wr a)
..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

Sharing and Racing

```
void loop(int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}

..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

- ➡ Operation are not instantaneous:
- ➡ Such an operation can becomes visible to different threads are different times
- ➡ apparent order of operations may not be *consistent*
- ➡ Even *consistent* operations can be *concurrent* (vary from execution to execution)
- ➡ Program must remain correct no matter the order

Sharing and Racing

```
void loop(int &a, int n)
{
      for(int i=0; i<n; i += 3)
            a++;
}

..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

- ➡ Operation are not instantaneous:
- ➡ Such an operation can becomes visible to different threads are different times
- ➡ apparent order of operations may not be *consistent*
- ➡ Even *consistent* operations can be *concurrent* (vary from execution to execution)
- ➡ Program must remain correct no matter the order

Data race: Concurrent RW or WW operations

Race condition: If variable order of concurrent operations affects correctness

Subodh Kumar

Share Occasionally

```
void loop(int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}           (Rd a; Add1; Wr a)
..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

```
     movl   (%rdi), %eax
     addl   $1, %eax
     movl   $0, %edx
.L3: movl   %eax, %ecx
     addl   $3, %edx
     addl   $1, %eax
     cmpl   %edx, %esi
     jg     .L3
     movl   %ecx, (%rdi)
```

Share Occasionally

```
void loop(volatile int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}          (Rd a; Add1; Wr a)
..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

```
     movl   (%rdi), %eax
     addl   $1, %eax
     movl   $0, %edx
.L3: movl   %eax, %ecx
     addl   $3, %edx
     addl   $1, %eax
     cmpl   %edx, %esi
     jg     .L3
     movl   %ecx, (%rdi)
```

Share Occasionally

```
void loop(volatile int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}                   (Rd a; Add1; Wr a)
..
int a = 0, n = 10;
#pragma omp parallel
   loop(a, n);
```

```
      movl    (%rdi), %eax
      addl    $1, %eax
      movl    $0, %edx
.L3:  movl    %eax, %ecx
      addl    $3, %edx
      addl    $1, %eax
      cmpl    %edx, %esi
      jg      .L3
      movl    %ecx, (%rdi)
```

```
.L3: movl   (%rdi), %eax
     addl   $1, %eax
     movl   %eax, (%rdi)
     addl   $3, %edx
     cmpl   %edx, %esi
     jg     .L3
     ret
```

Share Occasionally

```
void loop(volatile int &a, int n)
{
    for(int i=0; i<n; i += 3)
        a++;
}                (Rd a; Add1; Wr a)
..
int a = 0, n = 10;
#pragma omp parallel
    loop(a, n);
```

```
     movl   (%rdi), %eax
     addl   $1, %eax
     movl   $0, %edx
.L3: movl   %eax, %ecx
     addl   $3, %edx
     addl   $1, %eax
     cmpl   %edx, %esi
     jg     .L3
     movl   %ecx, (%rdi)
```

```
.L3: movl   (%rdi), %eax
     addl   $1, %eax
     movl   %eax, (%rdi)
     addl   $3, %edx
     cmpl   %edx, %esi
     jg     .L3
     ret
```

var = val need not be "seen"

or, need not be "atomically" seen

Review

- Task graphs and parallel programming models
- Shared memory programming
- Concurrency and race conditions
- Memory and Consistency