ŁUKASZ RACHWALSKI

ASYNCHRONOUS TASK PROCESSING

MOTIVATION

- Extend my application with asynchronous processing and:
    - Don’t want to care about threads management
    - Define number of threads and operations processed asynchronously
    - Have ability to get result of processed operations
    - When exit wait for all requested operations until they are done

PRESENTATION PLAN

- Queues
    - Blocking queue
- Task processors
    - Multi producer single consumer
    - Multi producer multi consumer

C++11 STUFF

- <future>
  - std::condition_variable
  - std::async
  - std::future
  - std::packaged_task
- <mutex>
  - std::mutex
  - std::unique_lock

WHY QUEUE?

- Need container that:
  - store tasks to be processed in sequence
  - provide efficient adding to the end and removing from the front
  - provide efficient access to first element

WE'VE GOT A QUEUE IN STL!

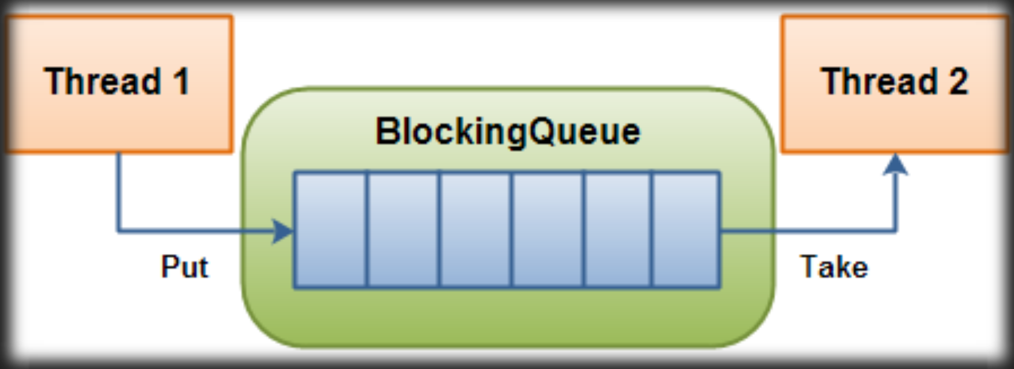
- std::queue - container adapter

```
template< class T,
    class Container = std::deque<T>
> class queue;
```

- Queue interface with constant complexity:
  - push - insert element to the end
  - pop - remove first element
  - front - access to first element
  - back - access to last element

WANT MORE…

- Thread safety
- Blocking
  - pop - blocks when queue is empty
  - push - blocks when queue is full



http://tutorials.jenkov.com

IS BLOCKING QUEUE NECESSARY?

- It is not necessary, but
  - it uses wait and wake mechanisms instead of busy waiting that simplifies implementation

LET'S MAKE A BLOCKING QUEUE

- Assumptions:
  - Parameterize type of holding elements and type of underlying container
  - Add limitation for number of elements holding on queue
  - „Push” function enqueue elements and blocks when queue is full until some element is popped or „Abort” function is called
  - „Pop” function removed first element from queue and blocks when is empty or „Abort” function is called
  - „Abort” terminates waiting „Pop” and „Push” threads through throwing exception

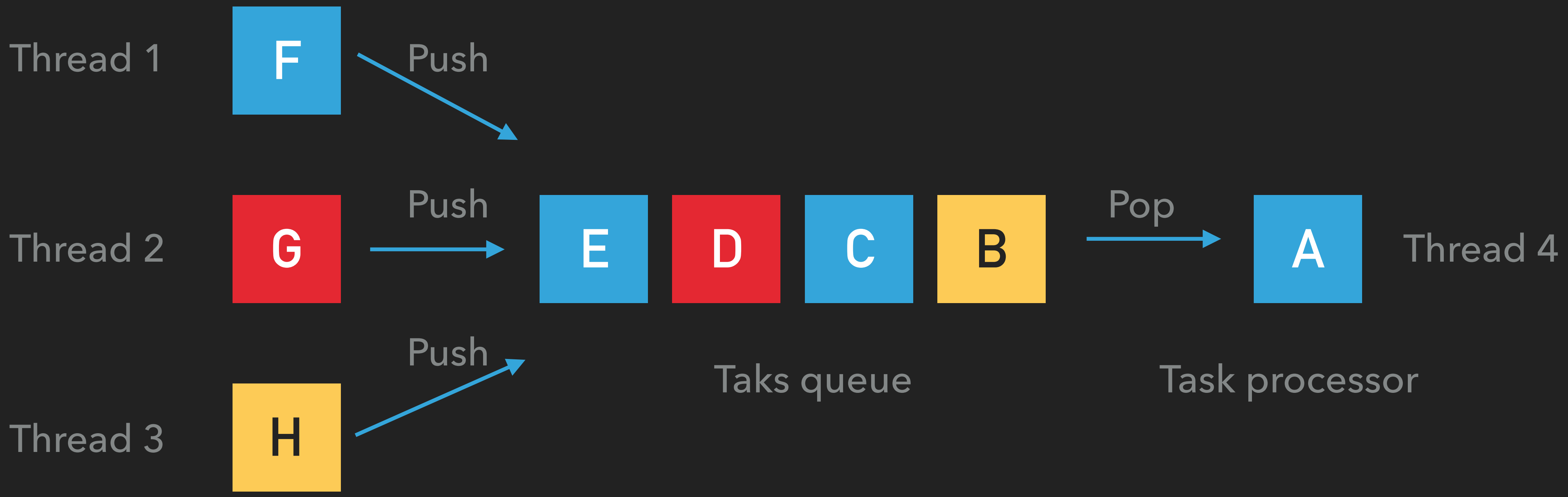
LET’S ASSUME THAT WE’VE GOT A BLOCKING QUEUE

TASK PROCESSORS

- Delegates task to be processed asynchronously
- Directly/indirectly performs thread management
- Variants:
  - Multi producer single consumer
  - Multi producer multi consumer (general case)

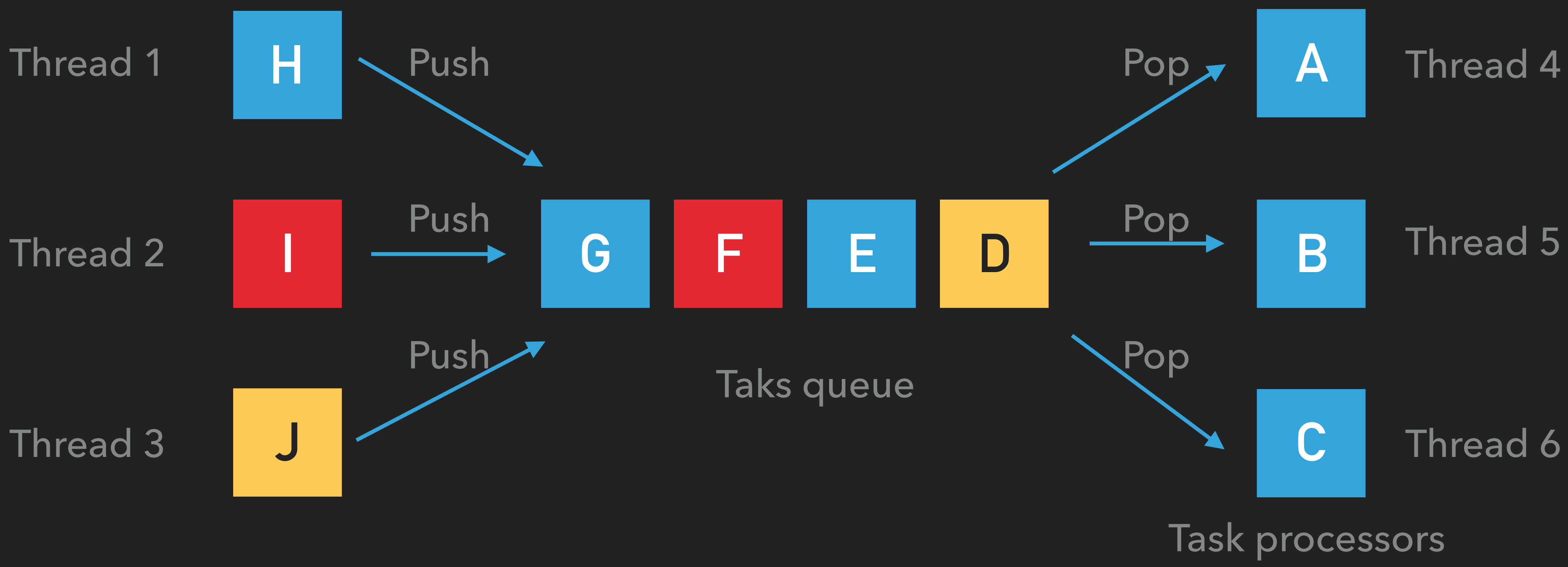
MULTI PRODUCERS SINGLE CONSUMER

- Multiple threads can push tasks to be perform asynchronously
- Single thread processes all tasks in FIFO order



MULTI PRODUCER MULTI CONSUMER

- Multiple threads can push tasks to be perform asynchronously
- Multiple threads process all tasks in FIFO order



LET'S MAKE A TASK PROCESSOR

- Assumptions:
  - Parameterize type of holding tasks and type of underlying queue
  - „Post” function enqueue tasks to be processed asynchronously

WORDS COUNTER EXAMPLE

- Problem
  - Read N books and find top 20 most common words.
- Solution
  - Parse all books asynchronously
  - Collect results and sort partially

QUESTIONS

THANK YOU