UNIT 4 · LINEAR STRUCTURES

15. Queue Applications

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

Agenda

S

Real Applications

Scheduling, spoolers, network buffers, resource management

P

Print Spooler Simulation

Watching the queue shrink from the front, job by job

B

Bank / Discrete Event Simulation

Modeling a waiting line without opening a real branch

T

Breadth-First / Level-Order

A queue drives level-by-level processing of a branching structure

D

Decision Framework

Choosing among array, linked list, stack, and queue

ANYWHERE FAIRNESS MATTERS

Real Applications of Queues

CPU Scheduling

An OS's ready queue: enqueue when a process becomes ready, dequeue when the CPU picks the next one (FCFS).

Print Spoolers

Jobs from multiple apps/users sent to the printer one at a time, in submission order.

Router / Network Buffers

Packets arriving faster than they can be forwarded are held in a queue, sized to available memory.

Resource Management

A DB connection pool, shared printers, a hospital's MRI machine — one resource, many requesters, fair order.

A DIRECT, EVERYDAY FIFO APPLICATION

Print Spooler

print_spooler.cpp

```
queue<PrintJob> spooler;
spooler.push({"Assignment1.pdf", 3});
spooler.push({"Timetable.docx", 1});
spooler.push({"LabManual.pdf", 12});

while (!spooler.empty()) {
    PrintJob job = spooler.front();
    spooler.pop();
    // ...print job.documentName
}
```

16 pages

printed in submission order across 3 jobs

Whichever job was submitted first is always at front() — a later job can never "jump the line," no matter how small.

WATCHING FIFO IN ACTION

The Spooler's Queue State, Step by Step

After all 3 jobs submitted

FRONT

`Assignment1.pdf` | `Timetable.docx` | `LabManual.pdf`

After printing Assignment1.pdf

FRONT

`Timetable.docx` | `LabManual.pdf`

After printing Timetable.docx

FRONT

`LabManual.pdf`

Queue empty

(nothing left)

Modeling a Bank Teller Line

bank_simulation.cpp

```
queue<int> customerLine;  // arrival minute
int arrivalTimes[] = {0, 2, 2, 5, 9};

while (!customerLine.empty()) {
    int arrival = customerLine.front();
    customerLine.pop();
    int serviceStart = max(arrival, teller1FreeAt);
    int waitTime = serviceStart - arrival;
    teller1FreeAt = serviceStart + 3;
}
```

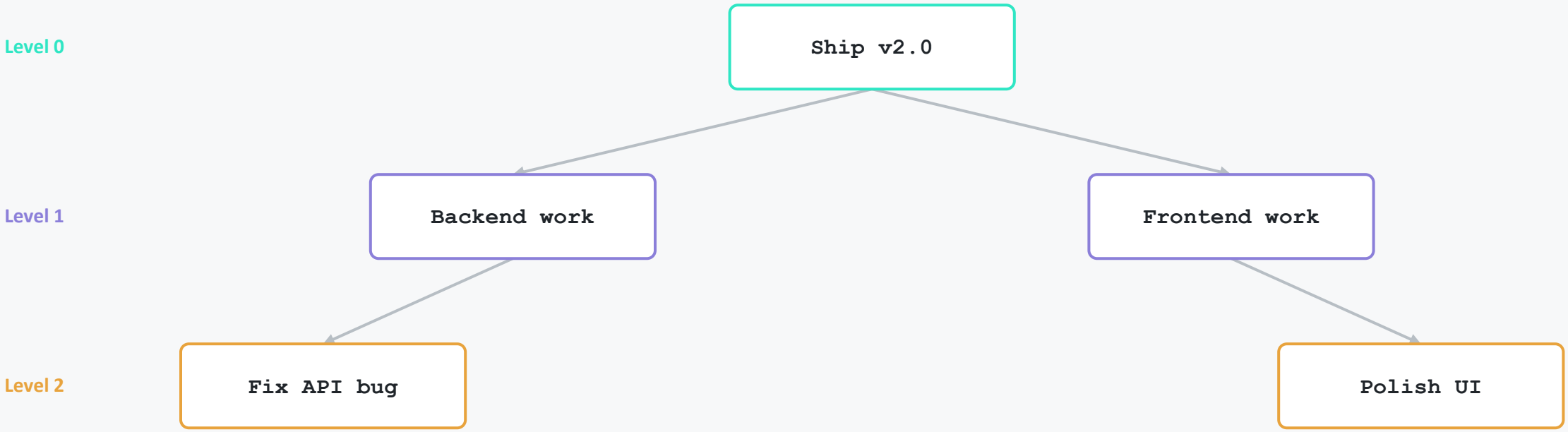
Cust.	Arrives	Served	Waited
1	0	0	0
2	2	3	1
3	2	6	4
4	5	9	4
5	9	12	3

2.4 min
average wait time, single teller

A SECOND, LESS OBVIOUS USE

Breadth-First / Level-Order Traversal

Push the root. Repeatedly dequeue a node, process it, and push its children onto the back of the same queue. FIFO guarantees every shallower node is processed before any deeper one.




```
Level 0: Ship v2.0     Level 1: Backend work, Frontend work     Level 2: Fix API bug, Polish UI
```

THE LINE THAT KEEPS LEVELS FROM BLEEDING

level_order_tasks.cpp

level_order_tasks.cpp

```
queue<TaskNode*> pending;
pending.push(root);
while (!pending.empty()) {
    int countAtThisLevel = pending.size();  // snapshot first!
    for (int i = 0; i < countAtThisLevel; i++) {
        TaskNode* current = pending.front();  pending.pop();
        if (current->left != nullptr) pending.push(current->left);
        if (current->right != nullptr) pending.push(current->right);
    }
}
```

Children are pushed onto the SAME queue mid-loop — snapshotting size() first is what keeps one level from bleeding into the next.

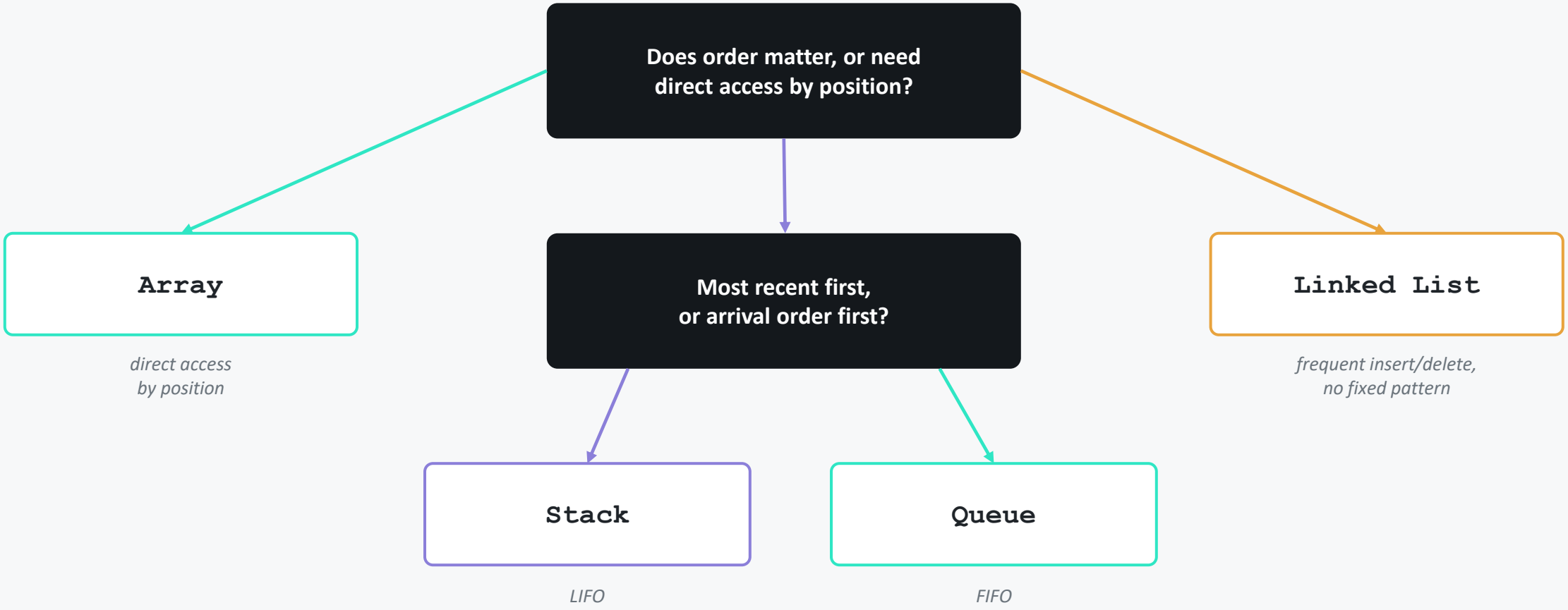
CLOSING UNITS 2–4

Selecting the Right Linear Structure

Array	Random access by index	Fast lookups by position, size roughly known
Linked List	Sequential, either direction	Frequent insert/delete, no fixed position
Stack	LIFO — one end only	Most-recent-first: undo, function calls, backtracking
Queue	FIFO — rear in, front out	Exact arrival order: scheduling, buffers

ASK ONE QUESTION

The Decision Flow



KEY TAKEAWAYS

Key Takeaways

- Queues model any system where multiple things wait their turn for one shared resource, in arrival order.
- Discrete event simulation is a real, widely-used application of queues — modeling behavior over time without building the real thing.
- A queue also drives breadth-first, level-order traversal: push the root, dequeue-and-enqueue-children, repeat.
- Snapshotting size() before each level's inner loop is what keeps levels from bleeding into each other.
- The choice among array, linked list, stack, and queue comes down to one question: what access pattern does the problem need?