Concurrency Patterns in Go

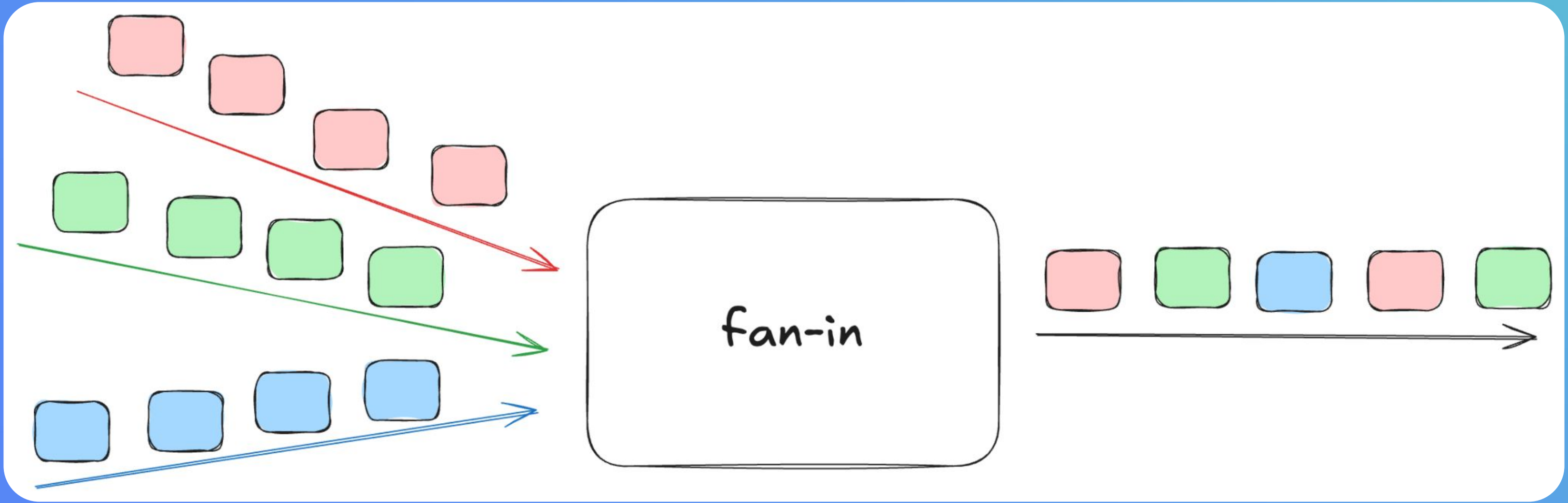
Artsiom Bukhautsou
Senior Backend Engineer @Nord Security

Agenda

1. Fan-in 🌱
2. Fan-out 🌱
3. Pipeline 🤔
4. Fan-in, Fan-out, Pipeline 🦾
5. Tee 🌱

Why learn these patterns?

Fan-in



Fan-in

Pros

- Simplifies aggregation
- Enhanced throughput

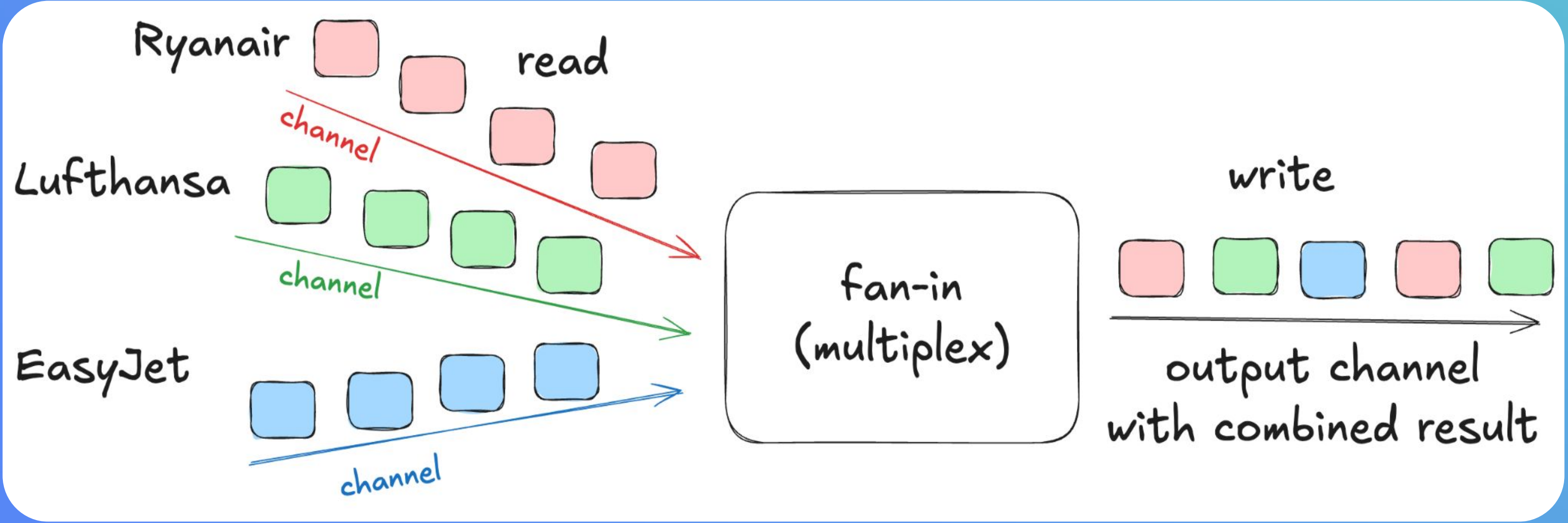
Cons

- Potential bottleneck

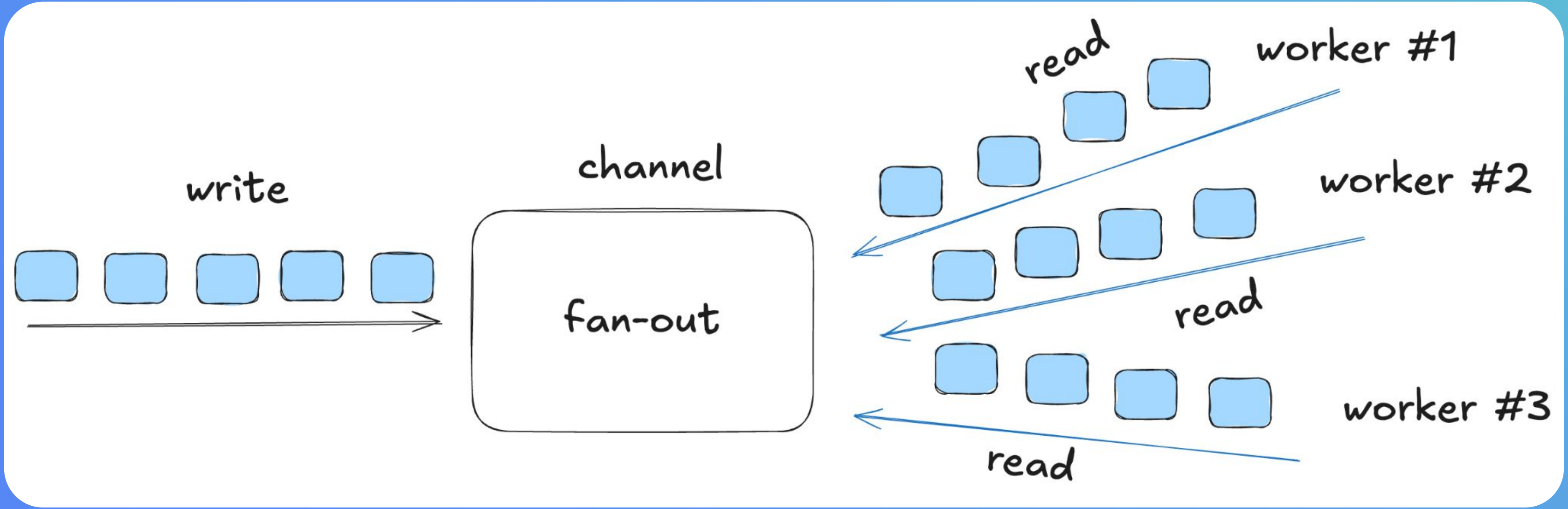
Use cases

- API response aggregation
- Log aggregation

Fan-in (Example)



Fan-out



read
worker #1
channel
write
worker #2
fan-out
read
worker #3
read

Fan-out

Pros

- Concurrent work distribution
- Improved throughput

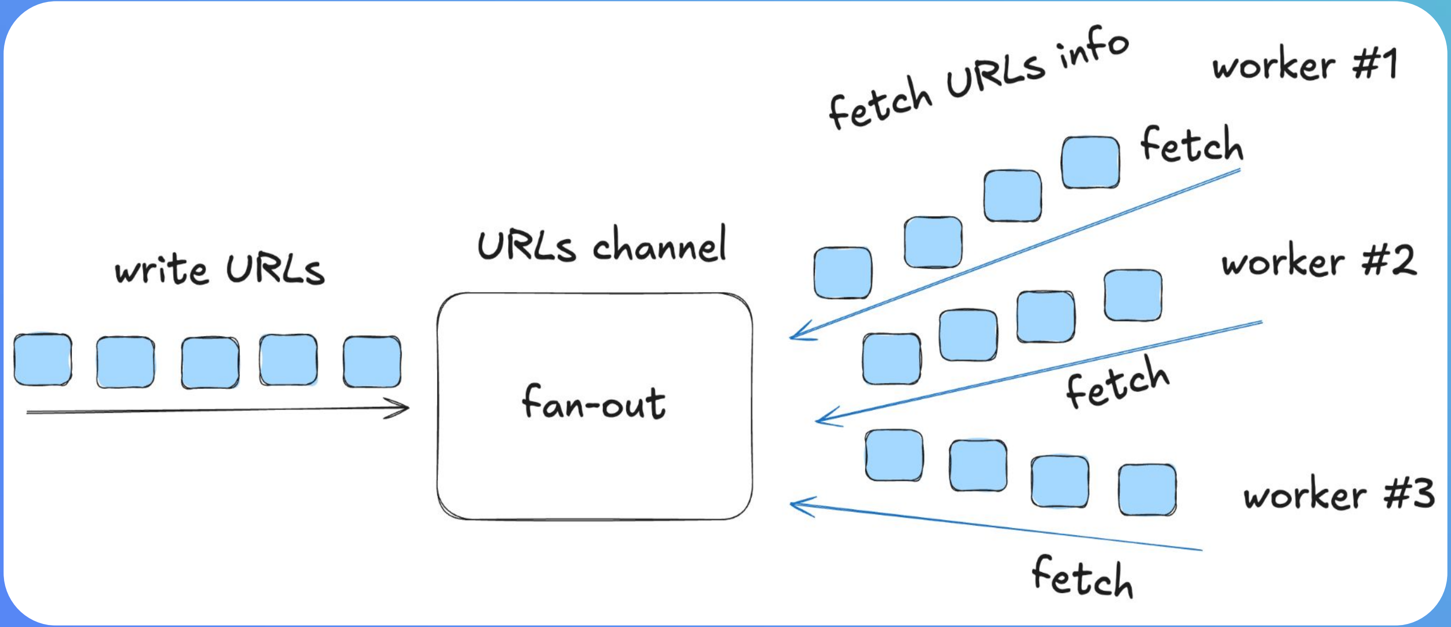
Cons

- Increased complexity

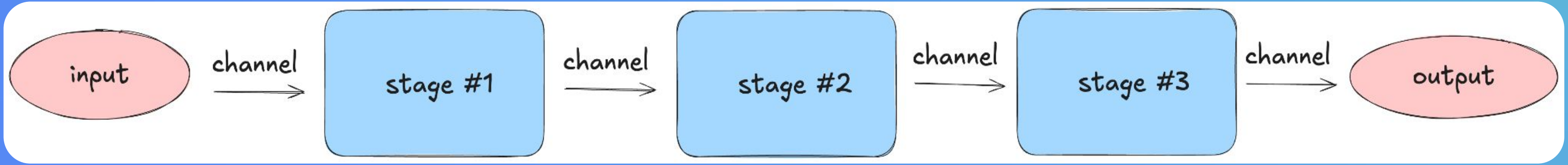
Use cases

- Web scraping
- Batch data processing

Fan-out (example)



Pipeline


input
channel
stage #1
channel
stage #2
channel
stage #3
channel
output

Pipeline

Pros

- Modular stages
- Concurrent stages

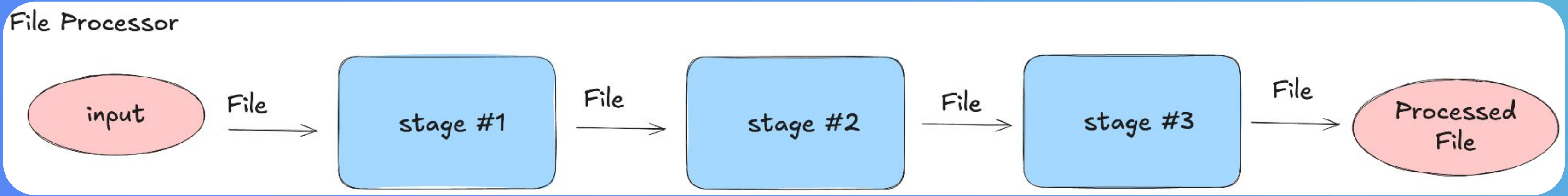
Cons

- Stage latency accumulation

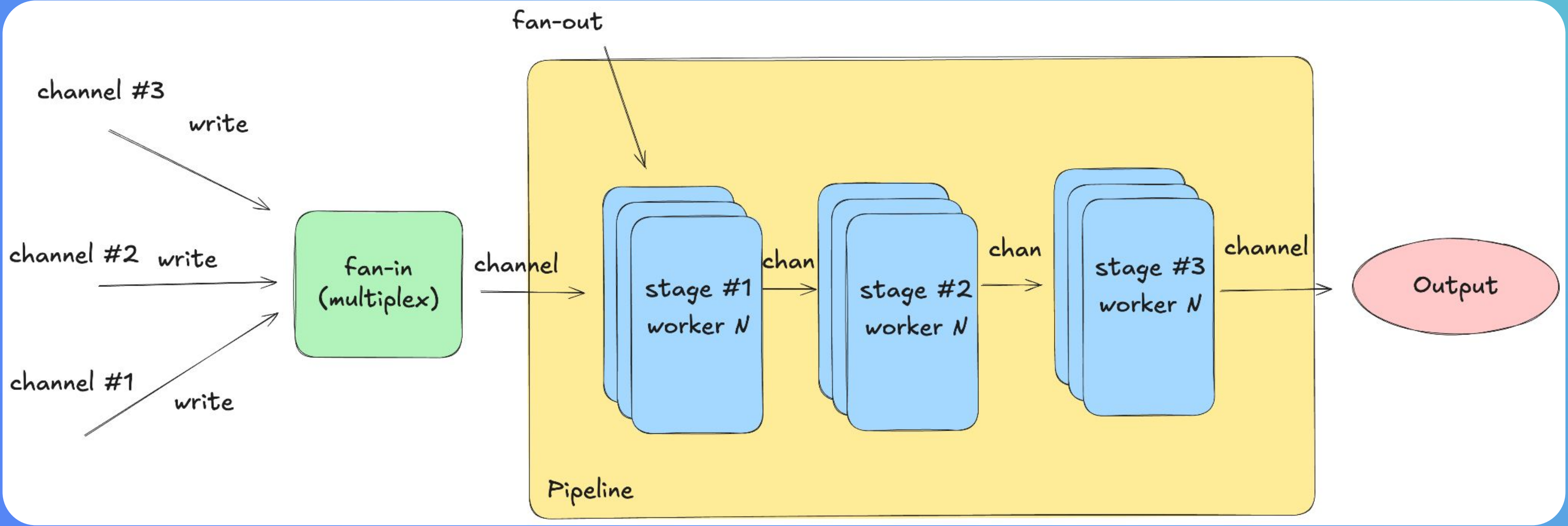
Use cases

- ETL processes
- Multimedia processing

Pipeline (Example)



Fan-in, Fan-out, Pipeline



Fan-in, Fan-out, Pipeline

Pros

- Maximized throughput
- Scalable architecture

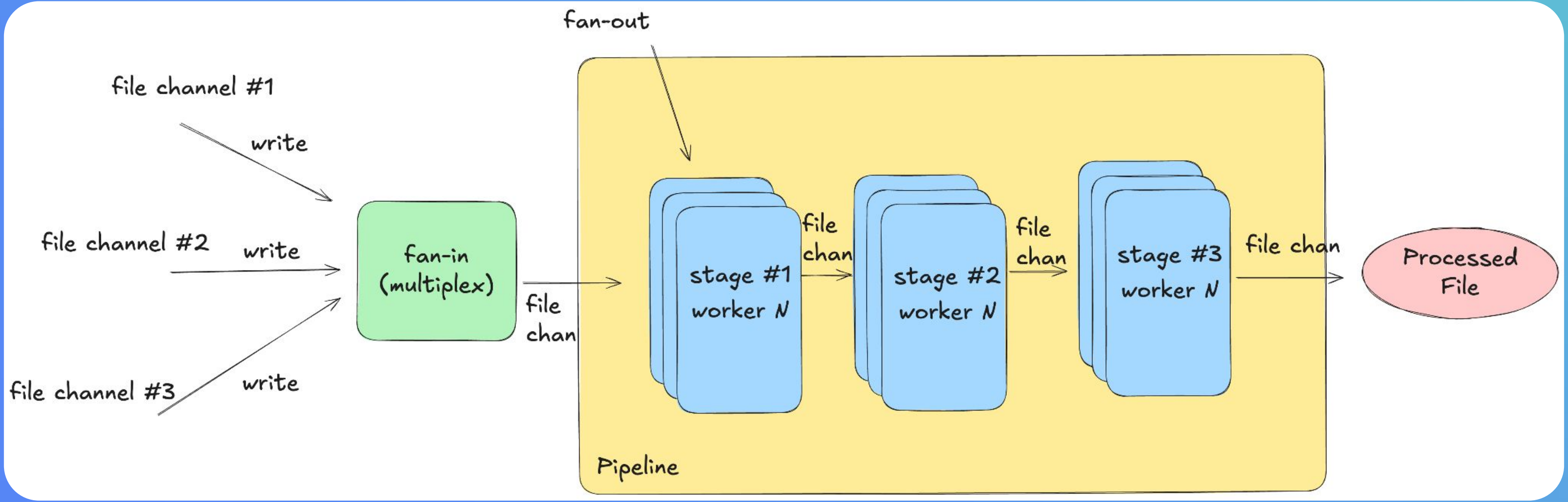
Cons

- System complexity

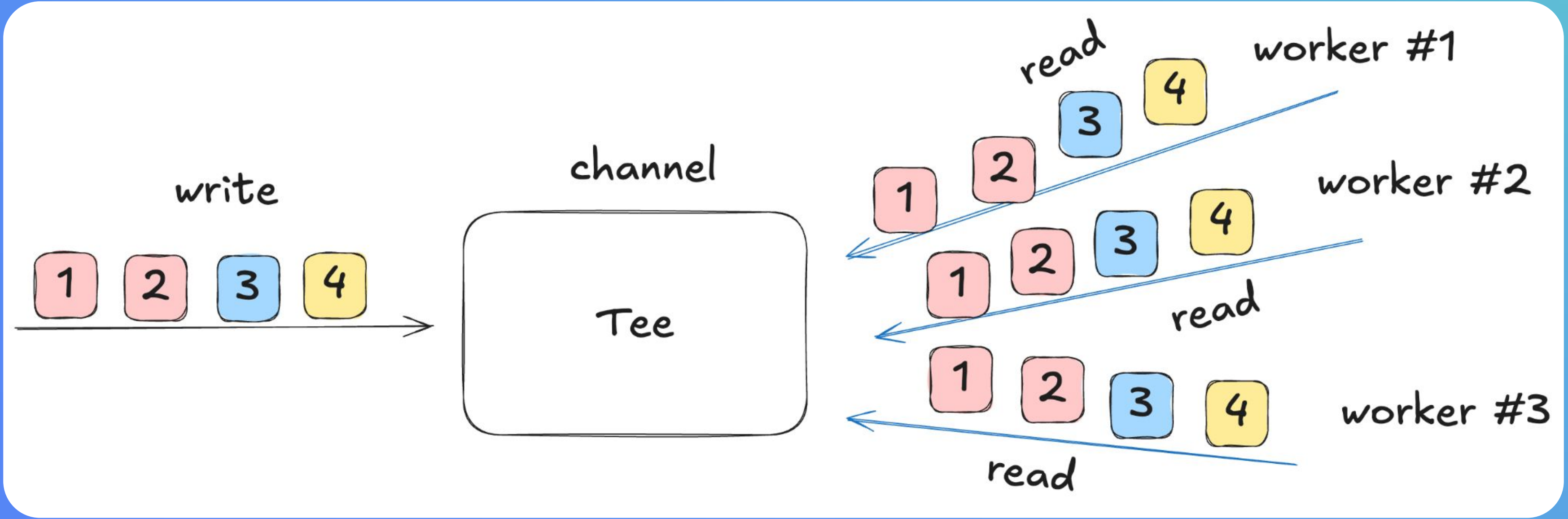
Use cases

- ETL processes
- Multimedia processing

Fan-in, Fan-out, Pipeline (Example)



Tee



write
1
2
3
4
channel
Tee
read
worker #1
1
2
3
4
worker #2
1
2
3
4
read
1
2
3
4
worker #3
read

Tee

Pros

- Efficient data distribution
- Decoupling consumers

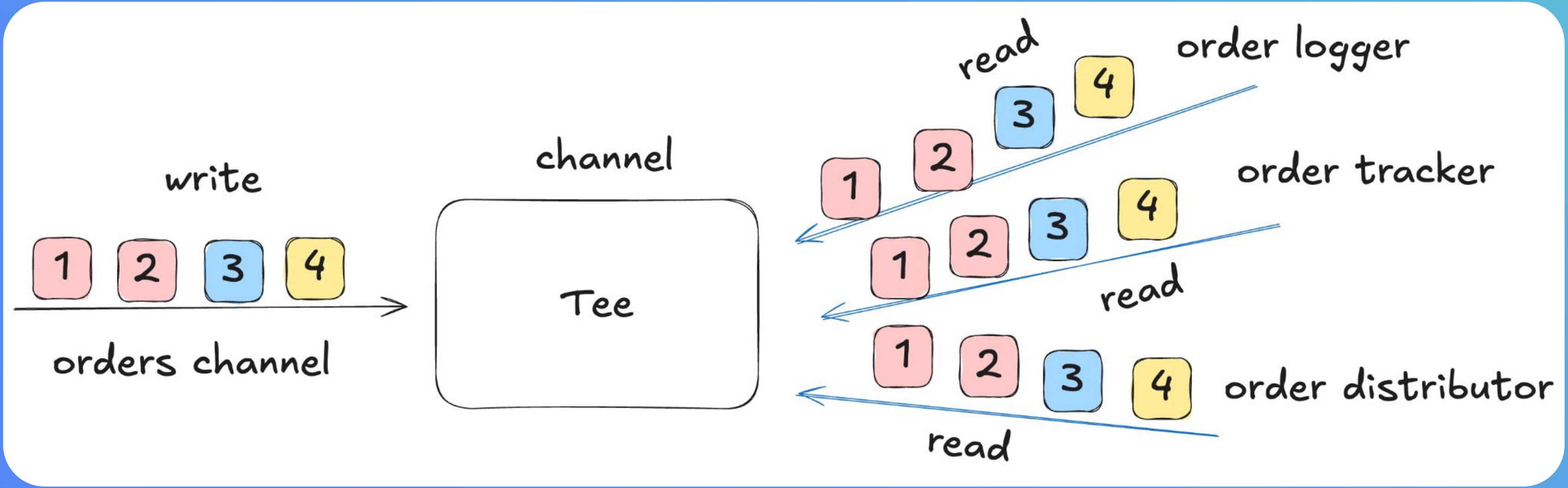
Cons

- No order guarantee

Use cases

- Logging and monitoring
- Broadcasting

Tee (Example)


read
order logger
4
3
2
1
channel
write
order tracker
1
2
3
4
Tee
read
orders channel
1
2
3
4
order distributor
read

NORD
SECURITY

My LinkedIn profile

Code examples

NORD
SECURITY

Thank you!