F# in the Real World

Yan Cui (@theburningmonk)

agenda

Hi, my name is Yan Cui.


gamesys

1 MILLION DAILY ACTIVE USERS

250 MILLION

Request Per Day

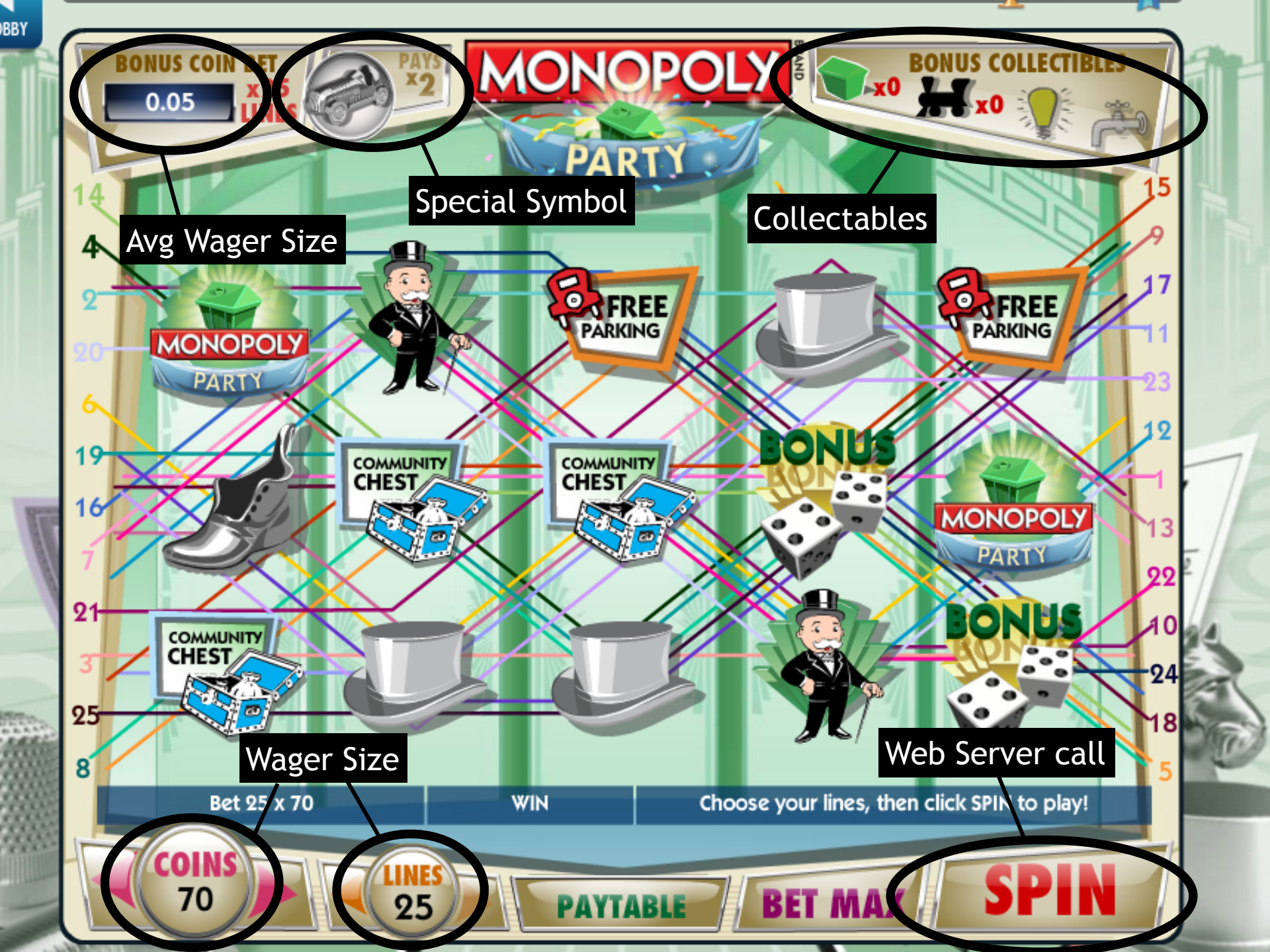
why F#?

time to market

correctness

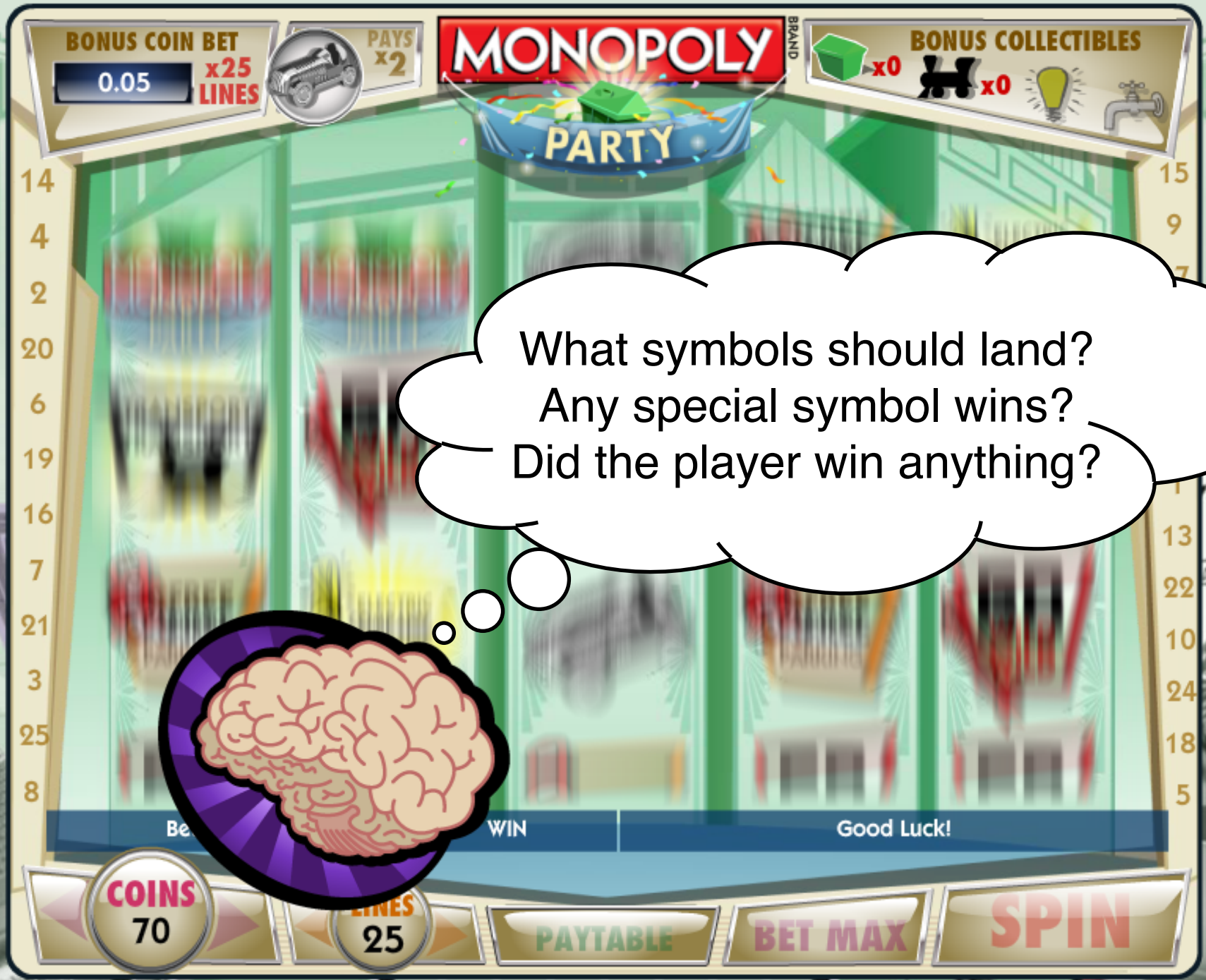
efficient

tame complexity


BONUS COIN BET
0.05
PAYS x2
MONOPOLY PARTY
BONUS COLLECTIBLES
x0
x0
Avg Wager Size
Special Symbol
Collectables
FREE PARKING
COMMUNITY CHEST
BONUS
Bet 25 x 70
WIN
Choose your lines, then click SPIN to play!
Wager Size
Web Server call
COINS 70
LINES 25
PAYTABLE
BET MAX
SPIN

- Line Win
  - X number of matching symbols on adjacent columns
  - Positions have to be a ‘line’
  - Wild symbols substitute for other symbols

- Scatter Win
  - X number of matching symbols anywhere
  - Triggers bonus game


BONUS COIN BET
0.05
x25 LINES
PAYS x2
MONOPOLY
PARTY
BONUS COLLECTIBLES
x0
x0
What symbols should land?
Any special symbol wins?
Did the player win anything?
WIN
Good Luck!
COINS
70
LINES
25
PAYTABLE
BET MAX
SPIN


BONUS COIN BET
0.05
x25 LINES
PAYS x2
MONOPOLY
PARTY
BONUS COLLECTIBLES
x0
x0
What the player's new average wager?
WIN
Good Luck!
COINS
70
LINES
25
PAYTABLE
BET MAX
SPIN


BONUS COIN BET
0.05
x25 LINES
PAYS x2
MONOPOLY
PARTY
BONUS COLLECTIBLES
x0
x0
Should the player receive collectables?
WIN
Good Luck!
COINS
70
LINES
25
PAYTABLE
BET MAX
SPIN

```
type Symbol = Standard   of string
            | Wild
```

```
type Symbol = Standard   of string
            | Wild
```

e.g. Standard “hat”

Standard “shoe”

Standard “bonus”

…

type Symbol = Standard of string

| Wild

i.e. Wild

```
type Win = LineWin    of int * Symbol * int
         | ScatterWin of Symbol * int
```

```
type Win = LineWin   of int * Symbol * int
         | ScatterWin  of Symbol * int
```

e.g. LineWin (5, Standard “shoe”, 4)

```
type Win = LineWin    of int * Symbol * int
         | ScatterWin  of Symbol * int
```

e.g. ScatterWin (Standard “bonus”, 3)

```
type LineNum = int
type Count   = int

type Win = LineWin   of LineNum * Symbol * Count
         | ScatterWin of Symbol * Count
```

```
type LineNum = int
type Count   = int

type Win = LineWin    of LineNum * Symbol * Count
         | ScatterWin of Symbol * Count
```

e.g. LineWin (5, Standard “shoe”, 4)

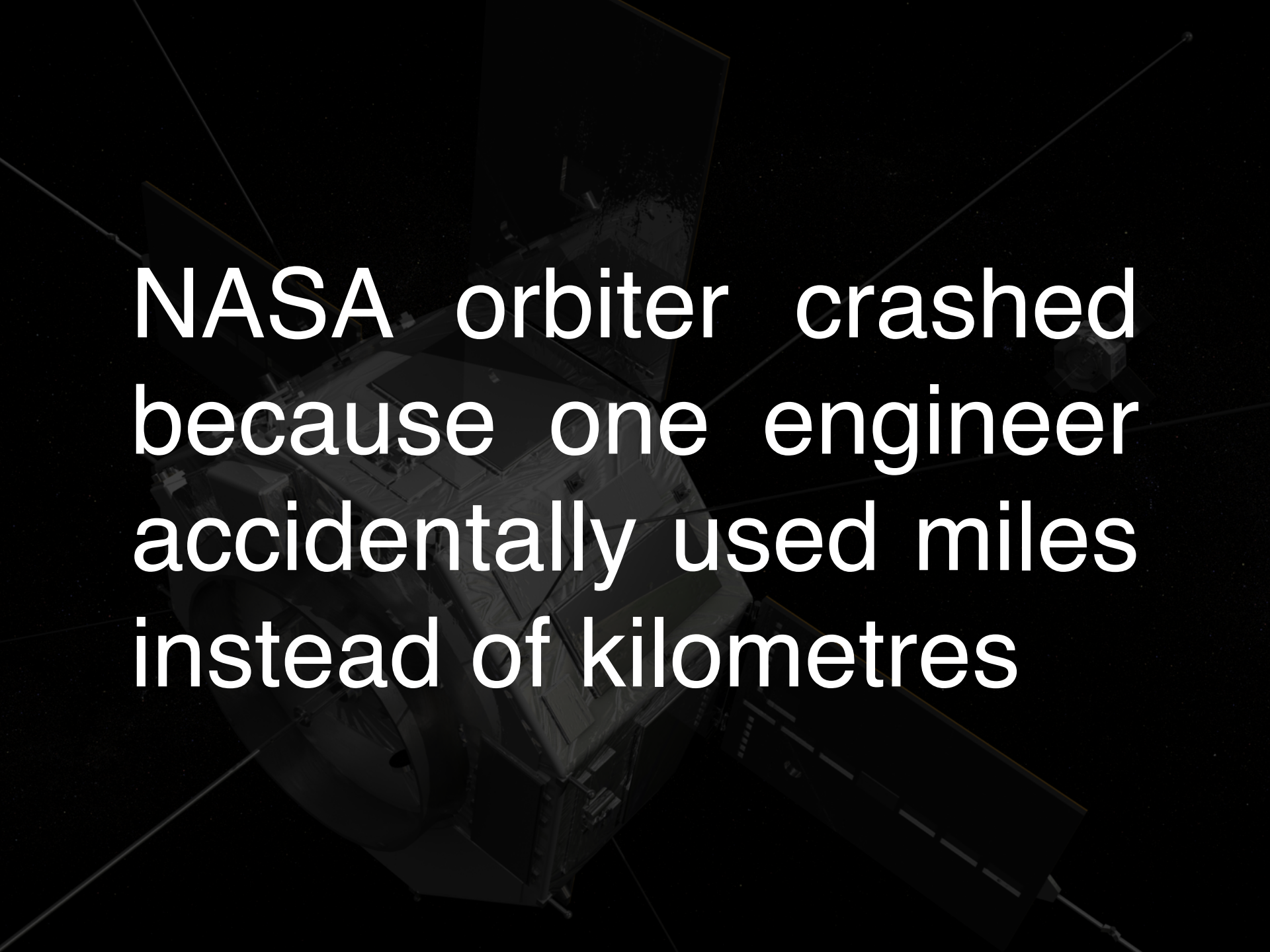
```
type LineNum = int
type Count   = int

type Win = LineWin   of LineNum * Symbol * Count
         | ScatterWin of Symbol * Count
```

e.g. ScatterWin (Standard “bonus”, 3)

make *invalid states*

unrepresentable


NASA orbiter crashed because one engineer accidentally used miles instead of kilometres


you’re never too smart
to make mistakes

```
[<Measure>]
type Pence
```

e.g. 42<Pence>

153<Pence>

…

```
10<Meter> / 2<Second>      = 5<Meter/Second>
10<Meter> * 2<Second>      = 20<Meter Second>
10<Meter> + 10<Meter>      = 20<Meter>
10<Meter> * 10             = 100<Meter>
10<Meter> * 10<Meter>      = 100<Meter^2>
10<Meter> + 2<Second>      // error
10<Meter> + 2              // error
```

```
10<Meter> / 2<Second>       = 5<Meter/Second>
10<Meter> * 2<Second>       = 20<Meter Second>
10<Meter> + 10<Meter>       = 20<Meter>
10<Meter> * 10              = 100<Meter>
10<Meter> * 10<Meter>       = 100<Meter^2>
10<Meter> + 2<Second>       // error
10<Meter> + 2               // error
```

```
type Wager      = int64<Pence>
type Multiplier = int

type Payout = Coins     of Wager
            | MultipliedCoins  of Multiplier * Wager
            | Multi     of Payout list
            | BonusGame
```

```
type Wager      = int64<Pence>
type Multiplier = int

type Payout = Coins     of Wager
            | MultipliedCoins  of Multiplier * Wager
            | Multi     of Payout list
            | BonusGame
```

```
type Wager      = int64<Pence>
type Multiplier = int

type Payout = Coins     of Wager
            | MultipliedCoins  of Multiplier * Wager
            | Multi     of Payout list
            | BonusGame
```

```
type State =
    {
        AvgWager          : Wager
        SpecialSymbol     : Symbol
        Collectables      : Map<Collectable, Count>
    }
```


BONUS COIN BET
70
x25 LINES
PAYS x2
MONOPOLY
PARTY
BONUS COLLECTIBLES
x0
x1
14
4
2
20
6
19
16
7
21
3
25
8
15
9
17
11
23
12
1
13
22
10
24
18
5
FREE PARKING
COMMUNITY CHEST
700
CHANCE
Bet 25 x 70
WIN 700.00
Line 17 wins 700.00
COINS
70
LINES
25
PAYTABLE
BET MAX
SPIN


BONUS COIN BET
70
PAYS x2
MONOPOLY
PARTY
BONUS COLLECTIBLES
x0
x1
New avg wager
Got a Collectable!
FREE PARKING
COMMUNITY CHEST
CHANCE
A pay line win!
700
Bet 25 x 70
WIN 700.00
Line 17 wins 700.00
COINS 70
LINES 25
PAYTABLE
BET MAX
SPIN


BONUS COIN BET
2.80
x25 LINES
PAYS x2
MONOPOLY
PARTY
BRAND
BONUS COLLECTIBLES
x19
x3
14
4
2
20
6
19
16
7
21
3
25
8
15
9
17
11
23
12
1
13
22
10
24
18
5
BON
MONOPOLY
PARTY
CHANCE
FREE PARKING
COMMUNITY CHEST
Bet 1 x 70
WIN
Good Luck!
COINS
70
LINES
1
PAYTABLE
BET MAX
SPIN


BONUS COIN BET
2.80
PAYS x2
MONOPOLY
BRAND
BONUS COLLECTIBLES
x19
x3
Bonus Game!
PARTY
BON
MONOPOLY
PARTY
CHANCE
Betting small reduces avg wager!
FREE PARKING
COMMUNITY CHEST
Bet 1 x 70
WIN
Good Luck!
COINS
70
LINES
1
PAYTABLE
BET MAX
SPIN


WELCOME!
You've made it to the BONUS!
Click on the glowing squares
to place your stations...
x1
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO


WELCOME!
You've made it to the BONUS!
Click on the glowing squares
to place your stations...
x1
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO

```
type MonopolyBoardSpace =
    | RailRoad                of bool
    | Property                of int
    | ElectricCompany         of bool
    | WaterCompany            of bool
    | Chance                  | CommunityChest
    | GotoJail                | Jail
    | FreeParking             | Go
```

```
type MonopolyBoardSpace =
    | RailRoad              of bool
    | Property              of int
    | ElectricCompany       of bool
    | WaterCompany          of bool
    | Chance                | CommunityChest
    | GotoJail              | Jail
    | FreeParking           | Go
```

```
type MonopolyBoardSpace =
    | RailRoad              of bool
    | Property              of int
    | ElectricCompany       of bool
    | WaterCompany          of bool
    | Chance          | CommunityChest
    | GotoJail        | Jail
    | FreeParking     | Go
```

```
type MonopolyBoardSpace =
    | RailRoad              of bool
    | Property              of int
    | ElectricCompany       of bool
    | WaterCompany          of bool
    | Chance                | CommunityChest
    | GotoJail              | Jail
    | FreeParking           | Go
```

```
[<Measure>]
type Position

type MonopolyBoard =
    {
        Spaces : MonopolyBoardSpace [ ]
    }
    member this.Item with
        get (pos : int<Position>) =
            this.Spaces.[int pos]
```

```
[<Measure>]
type Position

type MonopolyBoard =
    {
        Spaces : MonopolyBoardSpace [ ]
    }
    member this.Item with
        get (pos : int<Position>) =
            this.Spaces.[int pos]
```


GET OUT OF JAIL FREE
x0
GO WILD
210 x4
COMMUNITY CHEST
FREE PARKING
3 DOUBLES IN-A-ROW = GO TO JAIL
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO
ROLL
Bet 25 x 2.80
WIN
TOTAL WIN
1,680.00


Collected in the bonus game.
Gives player extra ‘lives’.
GET OUT OF JAIL FREE
x0
WILD
210 x4
And a pay line win!
3 DOUBLES IN-A-ROW = GO TO JAIL
Houses = multiplier on wins
GAME OVER
Coin size brought over from main game
COMMUNITY CHEST
FREE PARKING
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO
Bet 2 x 2.80
WIN
TOTAL WIN
1,680.00
ROLL

```
type BonusGameState =
    {
        Board       : MonopolyBoard
        Position    : int<Position>
        Lives       : int<Life>
        Doubles     : int<Double>
        CoinSize    : Wager
        TotalWin    : Wager
    }
```


Coin size brought over from main game
GET OUT OF JAIL FREE
x0
GO WILD
210 x4
COMMUNITY CHEST
FREE PARKING
3 DOUBLES IN-A-ROW = GO TO JAIL
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO
ROLL
Bet 2 x 2.80
WIN
TOTAL WIN
1,680.00

```
type BonusGameState =
    {
        Board       : MonopolyBoard
        Position    : int<Position>
        Lives       : int<Life>
        Doubles     : int<Double>
        CoinSize    : Wager
        TotalWin    : Wager
    }
```


GET OUT OF JAIL FREE
x0
GO
WILD
210 x4
COMMUNITY CHEST
FREE PARKING
3 DOUBLES IN-A-ROW = GO TO JAIL
And a pay line win!
Houses = multiplier on wins
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO
Bet 25 x 2.80
WIN
TOTAL WIN
1,680.00
ROLL

```
type BonusGameState =
    {
        Board       : MonopolyBoard
        Position    : int<Position>
        Lives       : int<Life>
        Doubles     : int<Double>
        CoinSize    : Wager
        TotalWin    : Wager
    }
```


GET OUT OF JAIL FREE
x0
GO WILD
210 x4
COMMUNITY CHEST
FREE PARKING
3 DOUBLES IN-A-ROW = GO TO JAIL
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
GO
Bet 25 x 2.80
WIN
TOTAL WIN
1,680.00
ROLL


GET OUT OF JAIL FREE
x0
CHANCE
3 DOUBLES IN-A-ROW = GO TO JAIL
COMMUNITY CHEST
Income tax refund. Collect 2.80
CONTINUE
The Fast-Dealing Property Trading
GO
Bet 25 x 2.80
WIN
TOTAL WIN
6,109.60


Collected in the bonus game.
Gives player extra ‘lives’.
GET OUT OF JAIL FREE
x0
3 DOUBLES IN-A-ROW = GO TO JAIL
MONOPOLY
The Fast-Dealing Property Trading Game
Bet 25 x 2.80
WIN
TOTAL WIN
1,680.00
ROLL

```
type BonusGameState =
    {
        Board       : MonopolyBoard
        Position    : int<Position>
        Lives       : int<Life>
        Doubles     : int<Double>
        CoinSize    : Wager
        TotalWin    : Wager
    }
```


GET OUT OF JAIL FREE
x0
GO
WILD
210 x4
COMMUNITY CHEST
FREE PARKING
3 DOUBLES IN-A-ROW = GO TO JAIL
MONOPOLY
The Fast-Dealing Property Trading Game
CHANCE
Bet 25 x 2.80
WIN
TOTAL WIN
1,680.00
ROLL
GO

0<Position>

let jail = 6<Position>

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
    …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | GotoJail when state.Lives = 0<Life> ->
            move jail state |> gameOver
    | GotoJail                    -> move jail state
    …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | RailRoad true              -> awardRailRoadMultiplier state
    | Property n when n > 0 -> awardPropertyMultiplier n state
    | ElectricityCompany true
    | WaterCompany true     -> awardUtilityMultiplier state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    …
    | CommunityChest      -> playCommunityChestCard state
    | Chance              -> playChanceCard state
   …
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    | RailRoad true              -> awardRailRoadMultiplier state
    | Property n when n > 0 -> awardPropertyMultiplier n state
    | ElectricityCompany true
    | WaterCompany true    -> awardUtilityMultiplier state
    | CommunityChest       -> playCommunityChestCard state
    | Chance               -> playChanceCard state
    | GotoJail when state.Lives = 0<Life> ->
          move jail state |> gameOver
    | GotoJail             -> move jail state
    | _                    -> state
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    | RailRoad true              -> awardRailRoadMultiplier state
    | Property n when n > 0 -> awardPropertyMultiplier n state
    | ElectricityCompany true
    | WaterCompany true    -> awardUtilityMultiplier state
    | CommunityChest       -> playCommunityChestCard state
    | Chance               -> playChanceCard state
    | GotoJail when state.Lives = 0<Life> ->
          move jail state |> gameOver
    | GotoJail             -> move jail state
    | _                    -> state
```

```
let rec move newPos state =
    match state.Board.[newPos] with
    | RailRoad true              -> awardRailRoadMultiplier state
    | Property n when n > 0 -> awardPropertyMultiplier n state
    | ElectricityCompany true
    | WaterCompany true    -> awardUtilityMultiplier state
    | CommunityChest       -> playCommunityChestCard state
    | Chance               -> playChanceCard state
    | GotoJail when state.Lives = 0<Life> ->
          move jail state |> gameOver
    | GotoJail             -> move jail state
    | _                    -> state
```


GET OUT OF JAIL FREE
CHANCE
CHANCE
3 DOUBLES IN-A-ROW = GO TO JAIL
CONGRATULATIONS!
TOTAL BONUS WIN
7,166.60
Bet 25 x 2.80
WIN
TOTAL WIN
7,166.60
GO
ROLL

Recap

lightweight syntax to create types and hierarchies

great for domain modelling

make invalid states unrepresentable

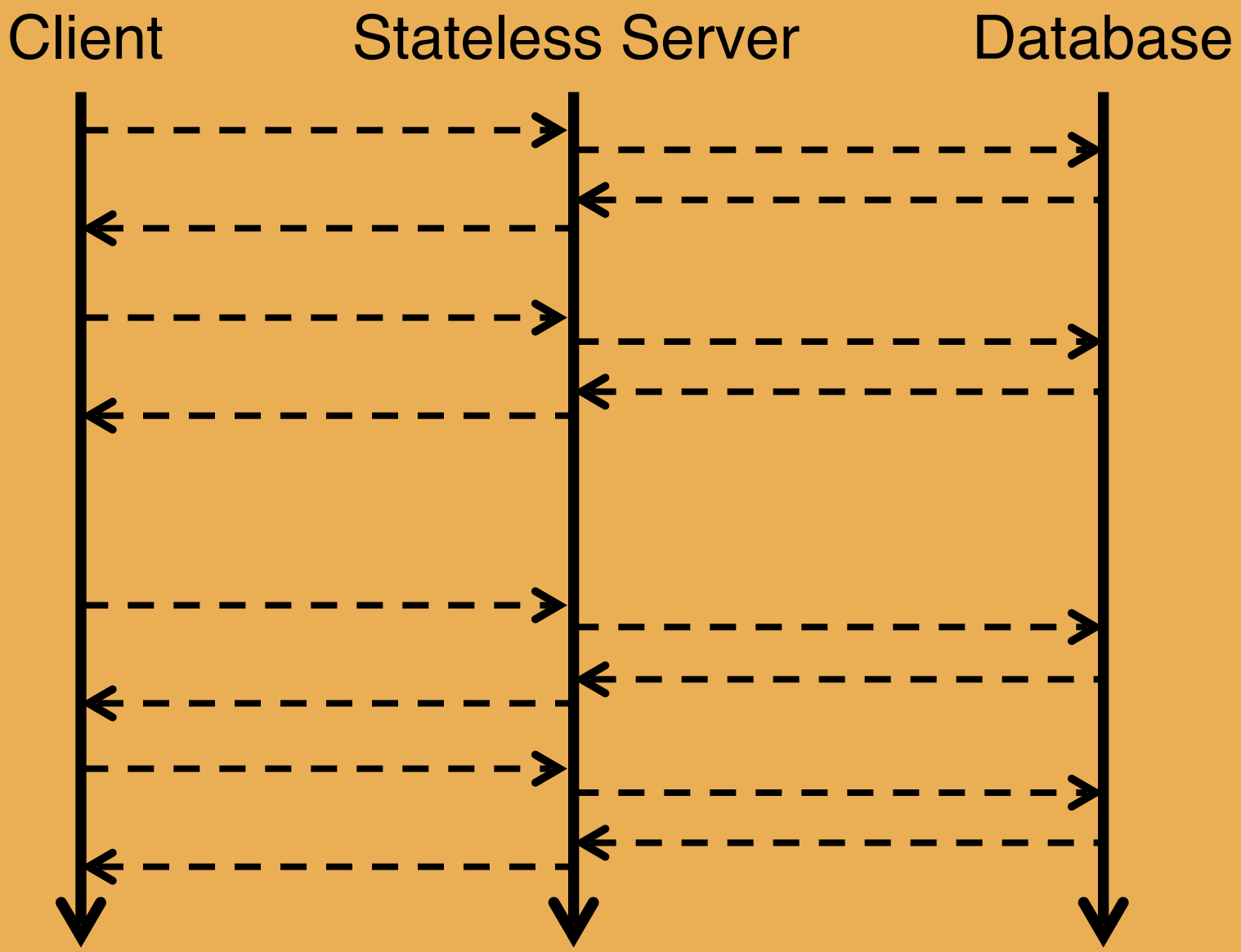
clear, concise way to handle branching

better correctness

order of *magnitude*
increase in productivity


Out of bait
Bashak, Trapper Captain
Your threats won't work on me, Dragon. Corrupted or no - you're not getting your claws on this. Quickly, Yan - come with me!
OK

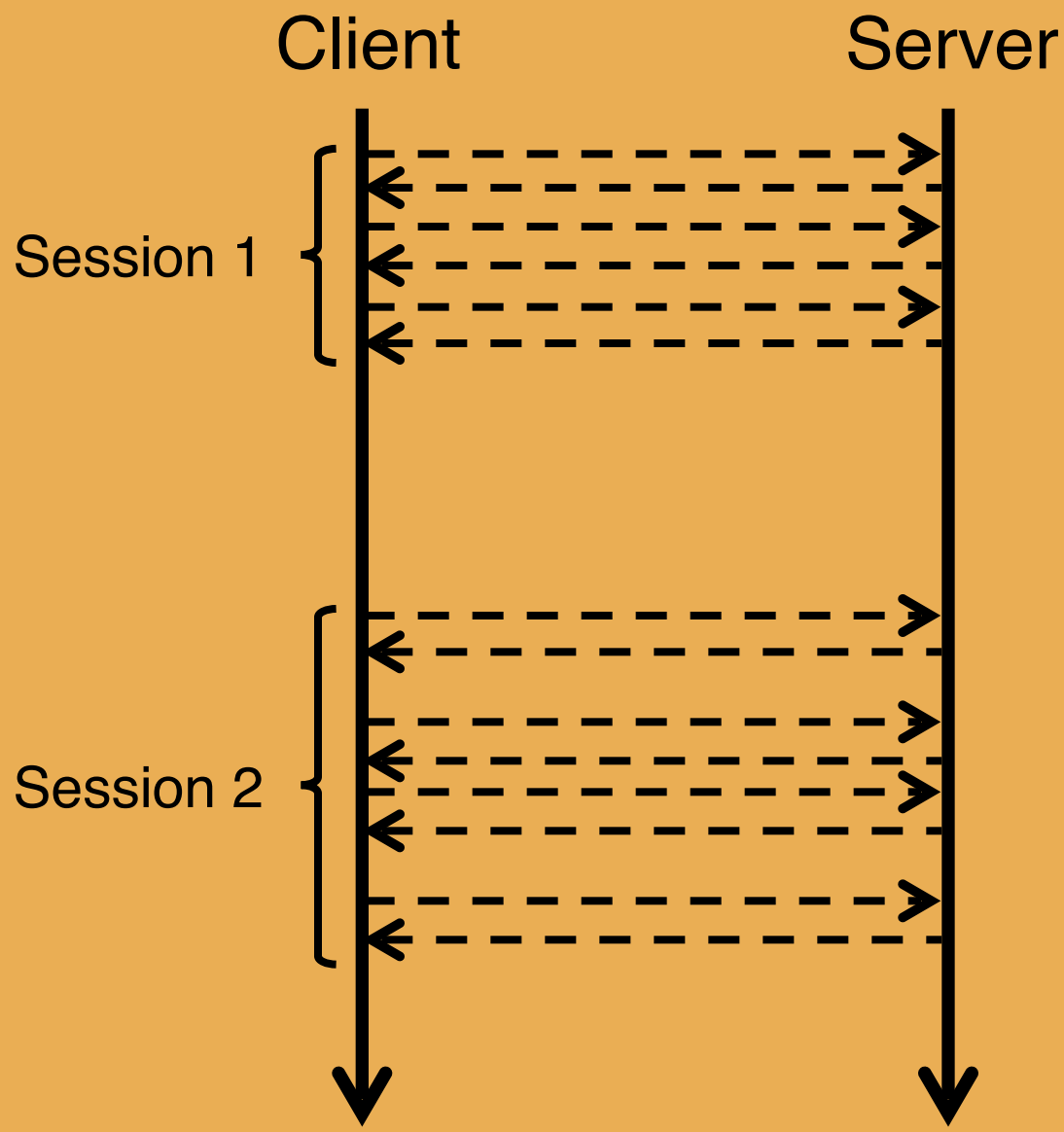
player states are big

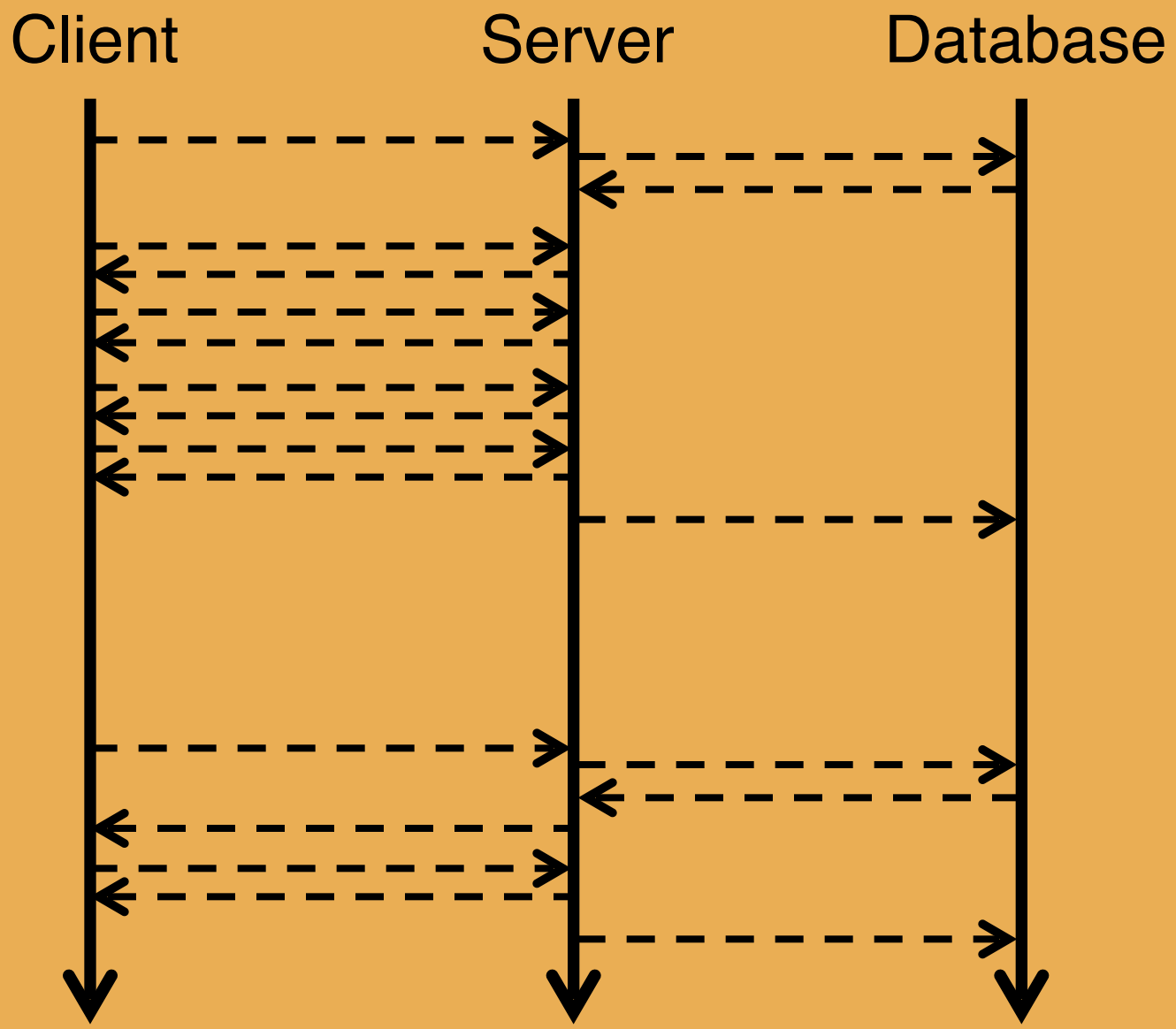

Client
Stateless Server
Database

1:1 read-write ratio

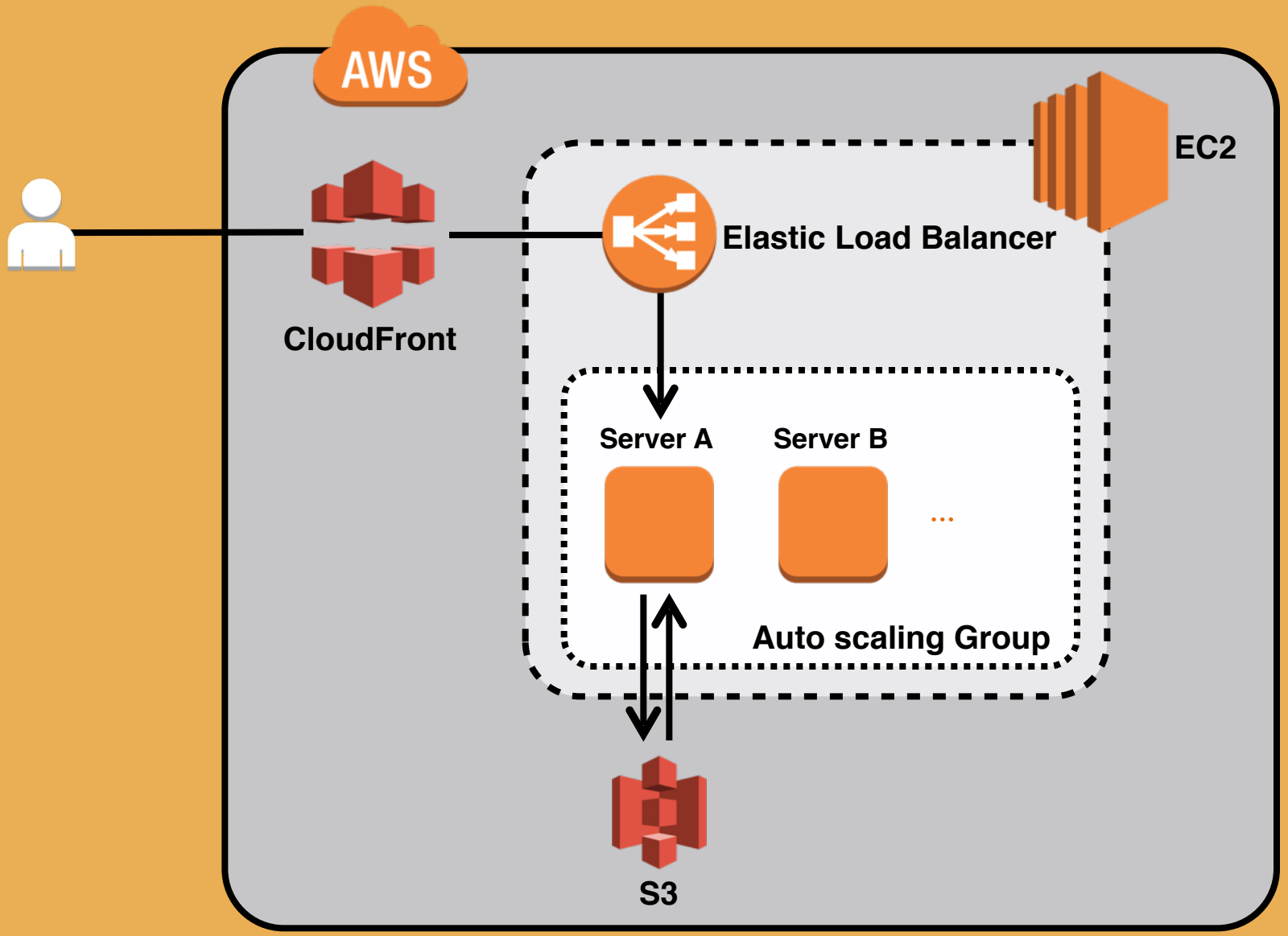
stateless =

Inefficient & Expensive


Client
Server
Session 1
Session 2


Client
Server
Database

Stateful Server



The Actor Model

An actor is the fundamental unit of computation which embodies the 3 things

- Processing
- Storage
- Communication

that are essential to computation.

-Carl Hewitt*

* http://bit.ly/HoNHbG

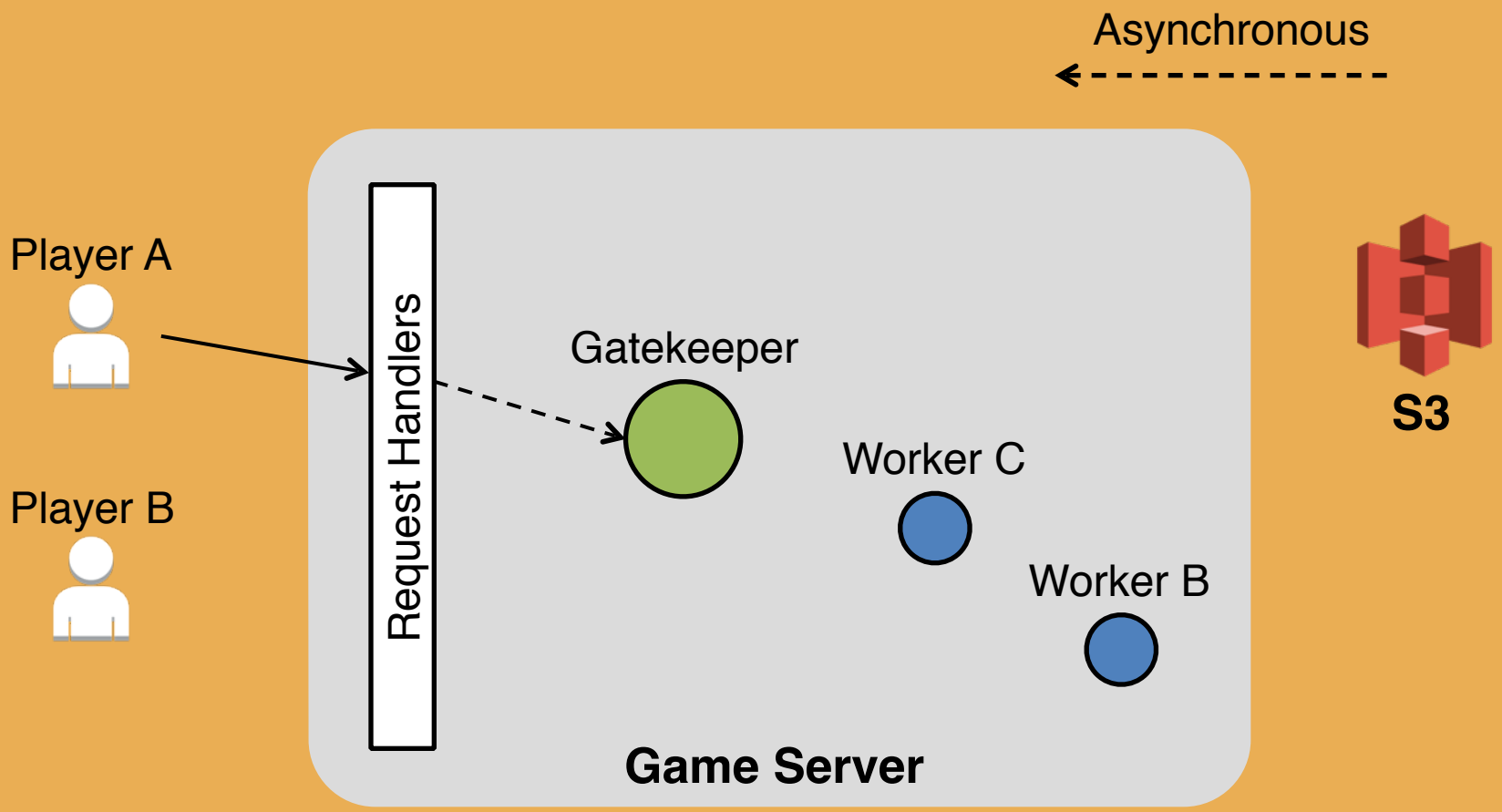
The Actor Model

- Everything is an actor
- An actor has a mailbox
- When an actor receives a message it can:
  - Create new actors
  - Send messages to actors
  - Designate how to handle the next message

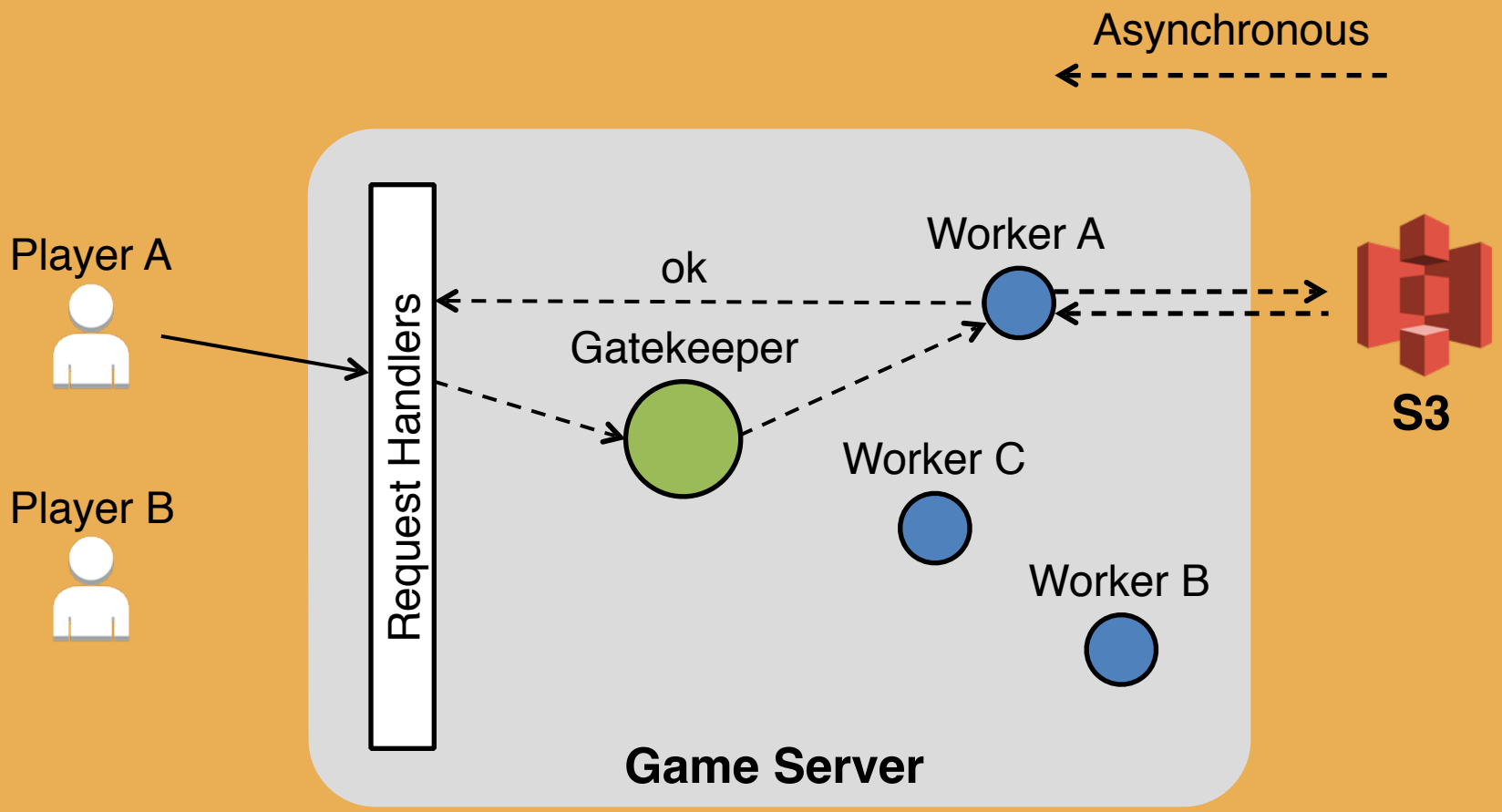
Stateful Server

- Gatekeeper
  - Manages the local list of active workers
  - Spawns new workers
- Worker
  - Manages the states for a player
  - Optimistic locking
  - Persist state after period of inactivity

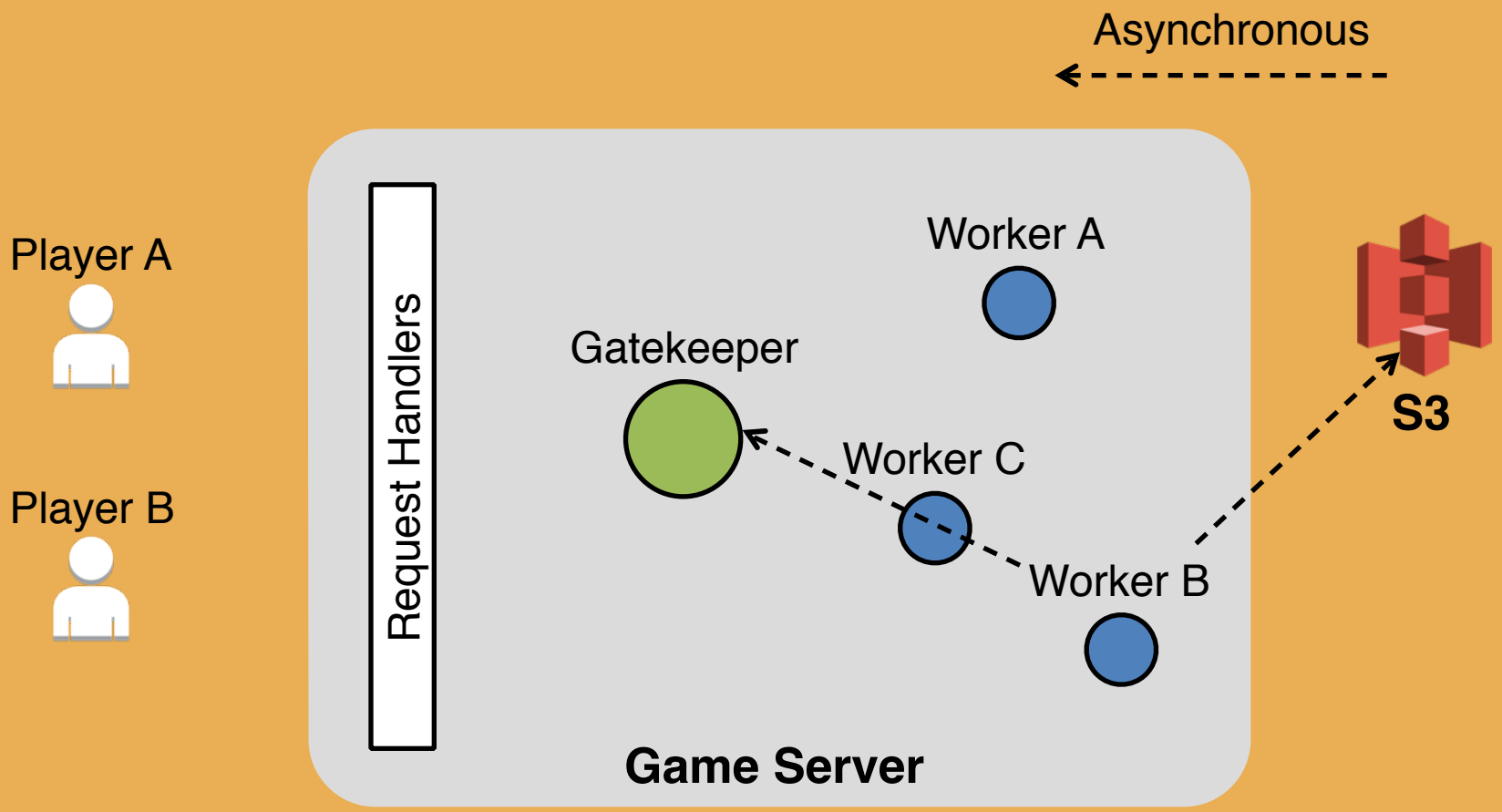
Stateful Server



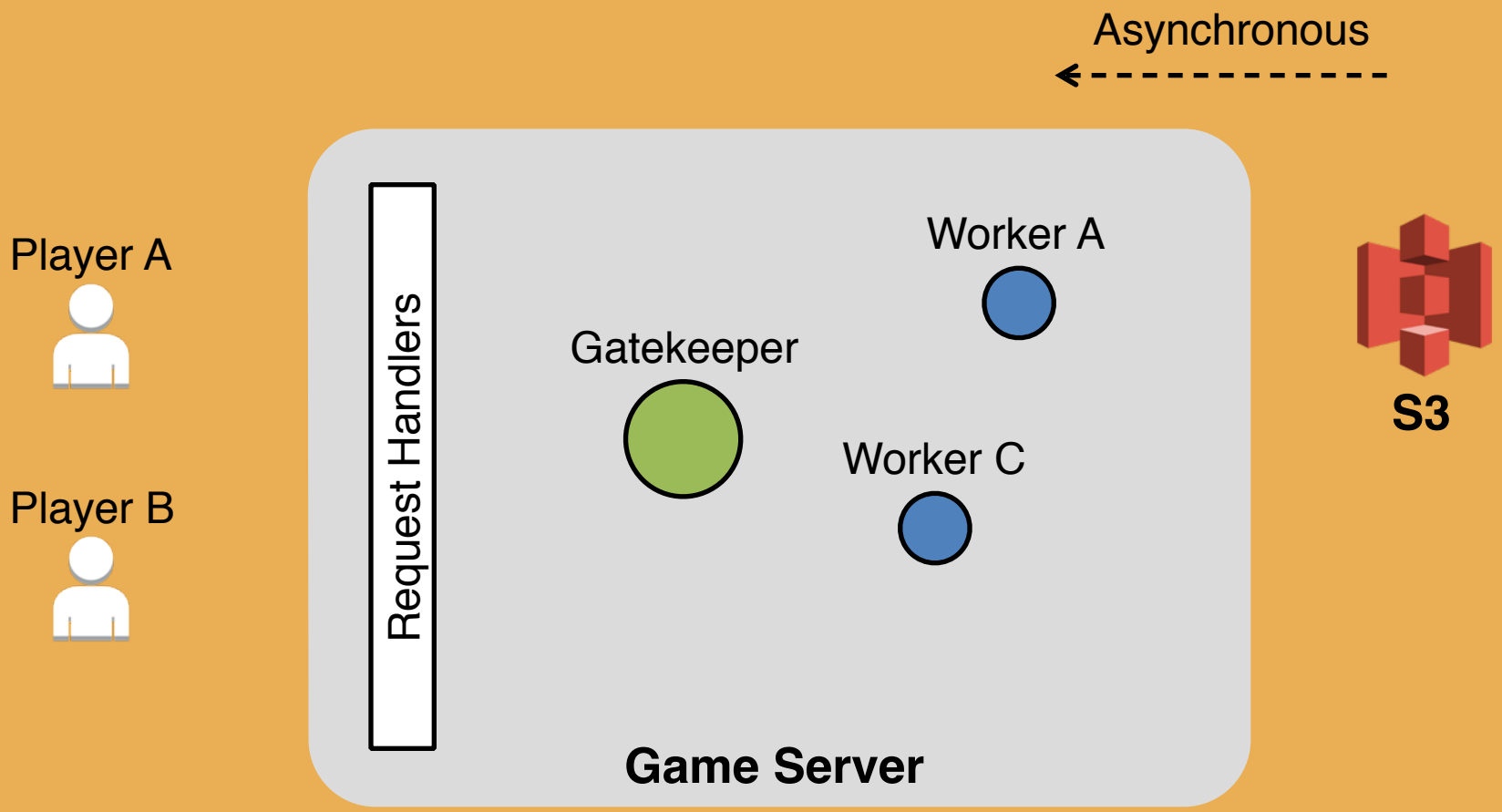
Stateful Server



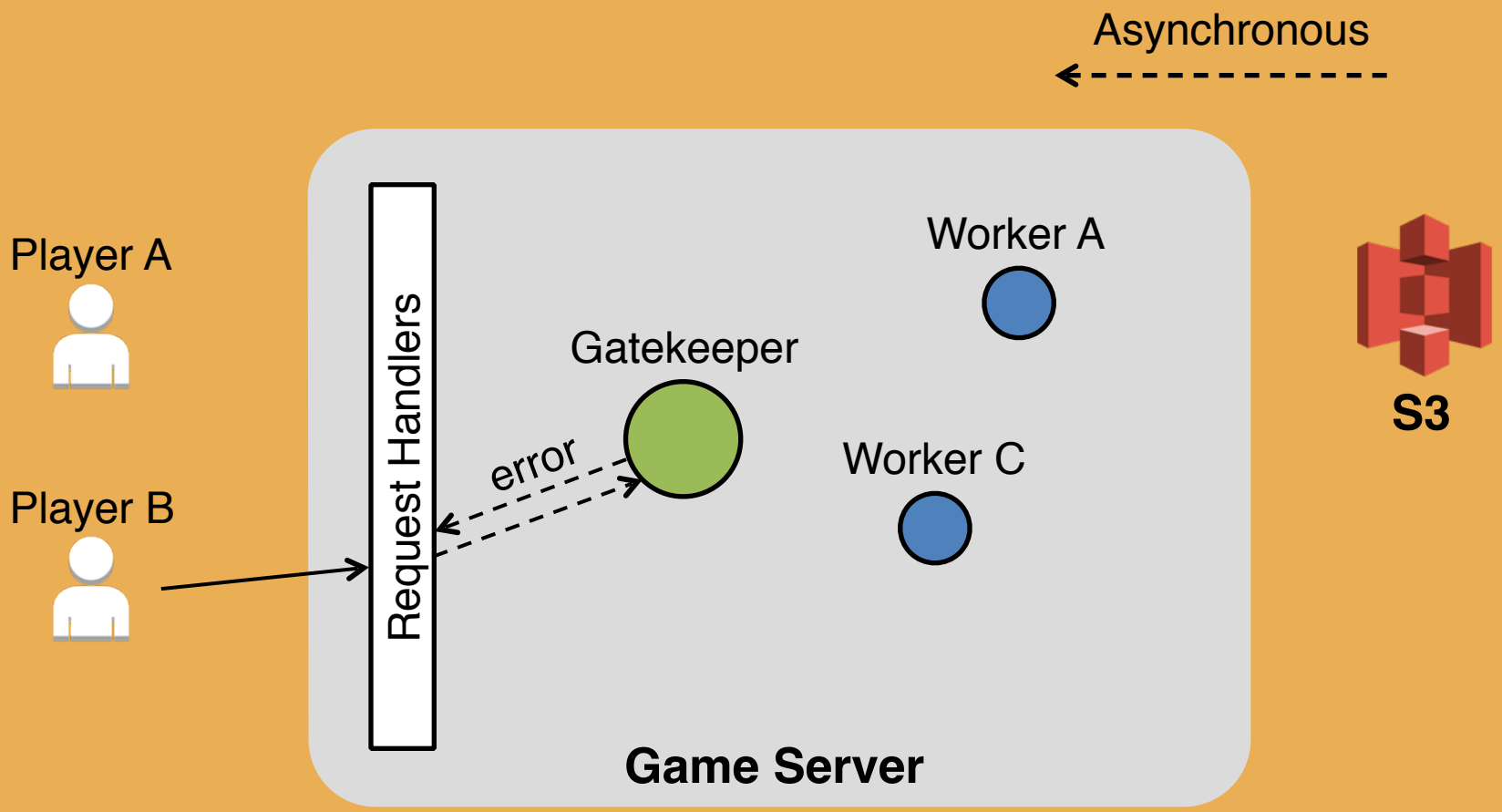
Stateful Server



Stateful Server



Stateful Server



```
type Agent<'T> = MailboxProcessor<'T>
```

```
type Agent<'T> = MailboxProcessor<'T>

type Message =
    | Get  of AsyncReplyChannel<...>
    | Put  of State * Version * AsyncReplyChannel<...>
```

```
type Agent<'T> = MailboxProcessor<'T>

type Message =
    | Get  of AsyncReplyChannel<...>
    | Put  of State * Version * AsyncReplyChannel<...>
```

```
type Result<'T> =
    | Success   of 'T
    | Failure   of Exception
```

```
type Result<'T> =
    | Success   of 'T
    | Failure   of Exception

type GetResult = Result<State * Version>
type PutResult = Result<unit>
```

```
type Agent<'T> = MailboxProcessor<'T>

type Message =
    | Get  of AsyncReplyChannel<GetResult>
    | Put  of State * Version * AsyncReplyChannel<PutResult>
```

```
type Agent<'T> = MailboxProcessor<'T>

type Message =
    | Get  of AsyncReplyChannel<GetResult>
    | Put  of State * Version * AsyncReplyChannel<PutResult>
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) =
            async { … }
        and closedState () =
            async { … }

        workingState (state, 1))
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) =
            async { … }
        and closedState () =
            async { … }

        workingState (state, 1))
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) =
            async { … }
        and closedState () =
            async { … }

        workingState (state, 1))
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) =
            async { … }
        and closedState () =
            async { … }

        workingState (state, 1))
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) =
            async { … }
        and closedState () =
            async { … }

        workingState (state, 1))
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None ->
            do! persist state
            return! closedState()
        …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None ->
            do! persist state
            return! closedState()
        …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None ->
            do! persist state
            return! closedState()
        …
    }
```

non-blocking I/O

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None ->
            do! persist state
            return! closedState()
        …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None ->
            do! persist state
            return! closedState()
        …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        …
        | Some(Get(reply)) ->
            reply.Reply <| Success(state, version)
            return! workingState(state, version)
        …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        …
        | Some(Put(newState, v, reply)) when version = v ->
            reply.Reply <| Success()
            return! workingState(newState, version+1)
      …
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        …
        | Some(Put(_, v, reply)) ->
            reply.Reply <| Failure(VersionMismatch(version, v))
            return! workingState(state, version)
    }
```

```
let rec workingState (state, version) =
    async {
        let! msg = inbox.TryReceive(60000)
        match msg with
        | None                                          -> …
        | Some(Get(reply))                              -> …
        | Some(Put(newState, v, reply)) when version = v -> …
        | Some(Put(_, v, reply))                        -> …
    }
```

```
and closedState () =
    async {
        let! msg = inbox.Receive()
        match msg with
        | Get(reply) ->
            reply.Reply <| Failure(WorkerStopped)
            return! closedState()
        | Put(_, _, reply) ->
            reply.Reply <| Failure(WorkerStopped)
            return! closedState()
    }
```

```
and closedState () =
    async {
        let! msg = inbox.Receive()
        match msg with
        | Get(reply) ->
            reply.Reply <| Failure(WorkerStopped)
            return! closedState()
        | Put(_, _, reply) ->
            reply.Reply <| Failure(WorkerStopped)
            return! closedState()
    }
```

```
type Worker (playerId) =
    let agent = Agent<Message>.Start(fun inbox ->
        let state = getCurrentState playerId

        let rec workingState (state, version) = async {
                let! msg = inbox.TryReceive(60000)  // timeout after 1 minute
                match msg with
                | None
                    -> do! persistState state
                       return! closedState()
                | Some(Get(reply))
                    -> Success(state, version) |> reply.Reply
                       return! workingState(state, version)
                | Some(Put(state', version', reply)) when version' = version
                    -> Success() |> reply.Reply
                       return! workingState(state', version + 1UL)
                | Some(Put(_, version', reply))
                    -> Failure(VersionMismatch(version', version)) |> reply.Reply
                       return! workingState(state, version)
            }
        and closedState () = async {
                let! msg = inbox.Receive()
                match msg with
                | Get(reply)       -> Failure(WorkerStopped) |> reply.Reply
                                      return! closedState()
                | Put(_, _, reply) -> Failure(WorkerStopped) |> reply.Reply
                                      return! closedState()
            }

        // start the agent in the working state
        workingState (state, 1UL))

    /// unit -> Async<Result<State * Version>>
    member this.GetState () = agent.PostAndAsyncReply(fun reply -> Get(reply))

    /// State * Version -> Async<Result<unit>>
    member this.PutState (state, version) = agent.PostAndAsyncReply(fun reply -> Put(state, version, reply))
```

5x efficient improvement

60% latency drop

no databases

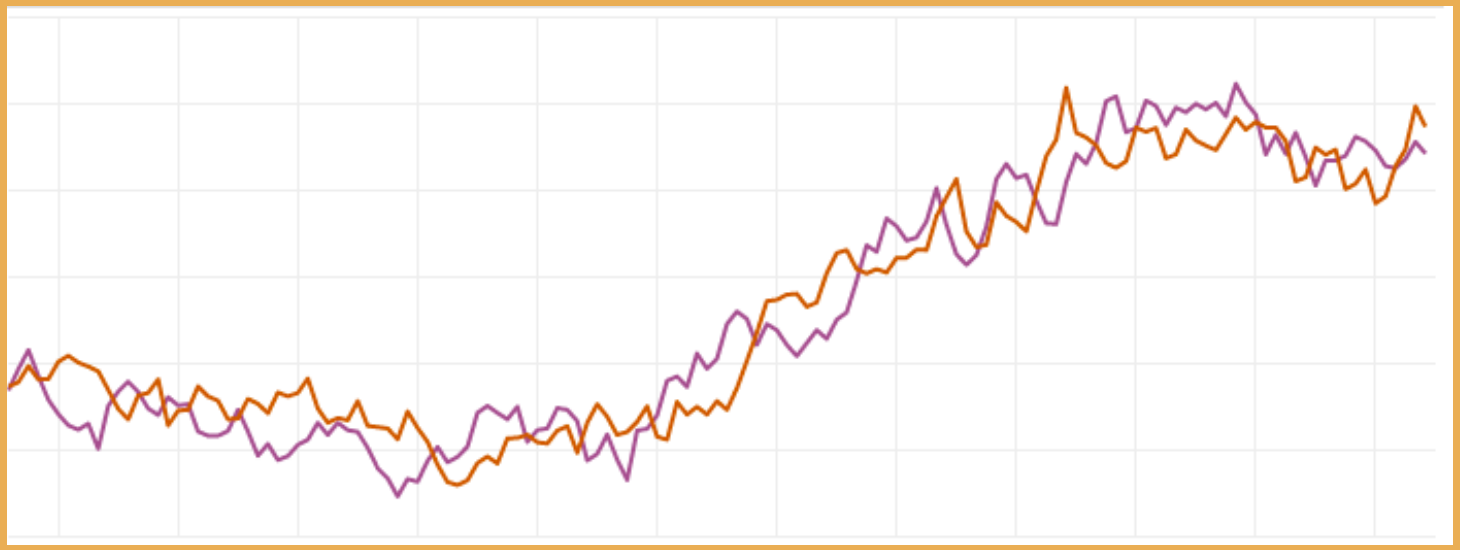
90+% cost saving

need to ensure server affinity

need to balance load

need to avoid hotspots

- Persist player state after short inactivity
- Move player to another server after persistence



need to gracefully scale down

Recap

Agents

- no locks
- async message passing

Agents

- no locks
- async message passing
- self-contained
- easy to reason with

Agents

- low level
- exceptions kills agents
- primitive error monitoring
- no supervision
- no distribution

akka.Net

MS Orleans

Cricket


ERLANG


elixir

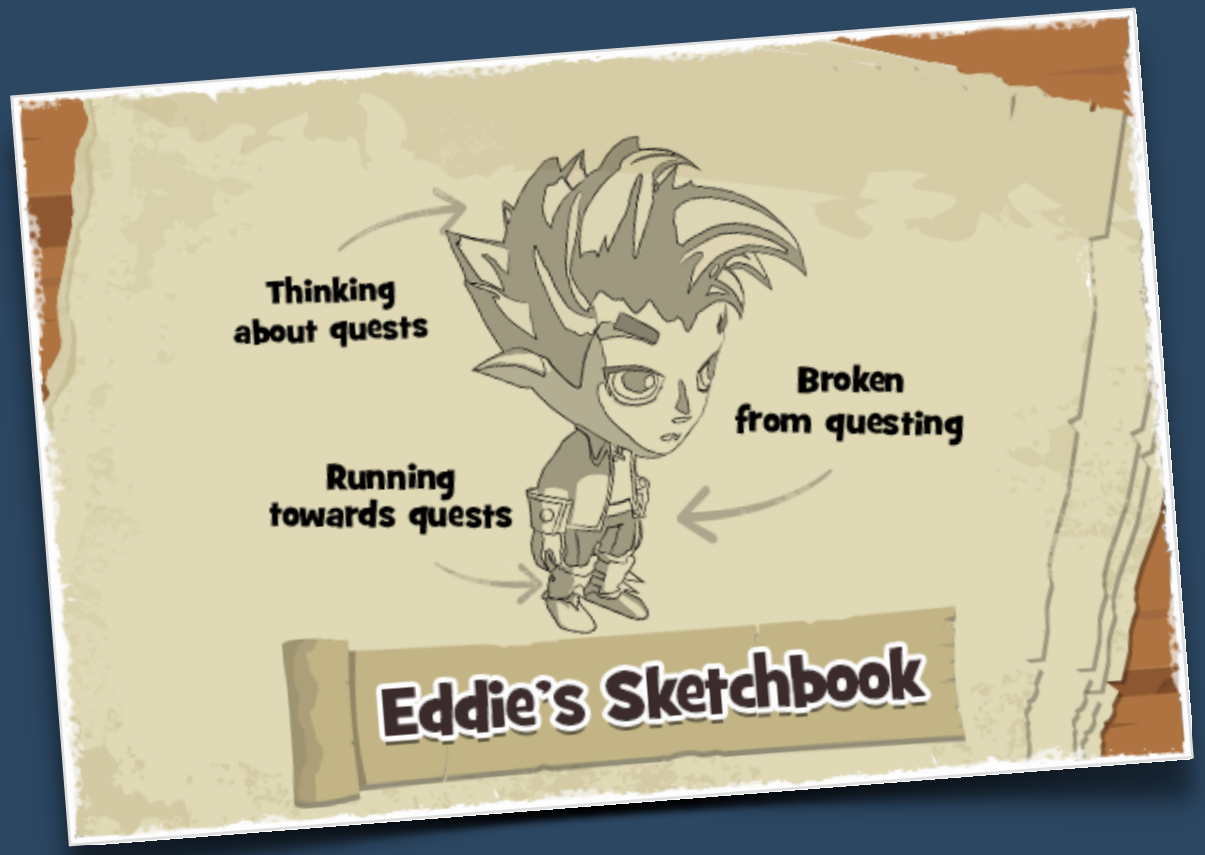

LISP
flavored
ERLANG

Async Workflow

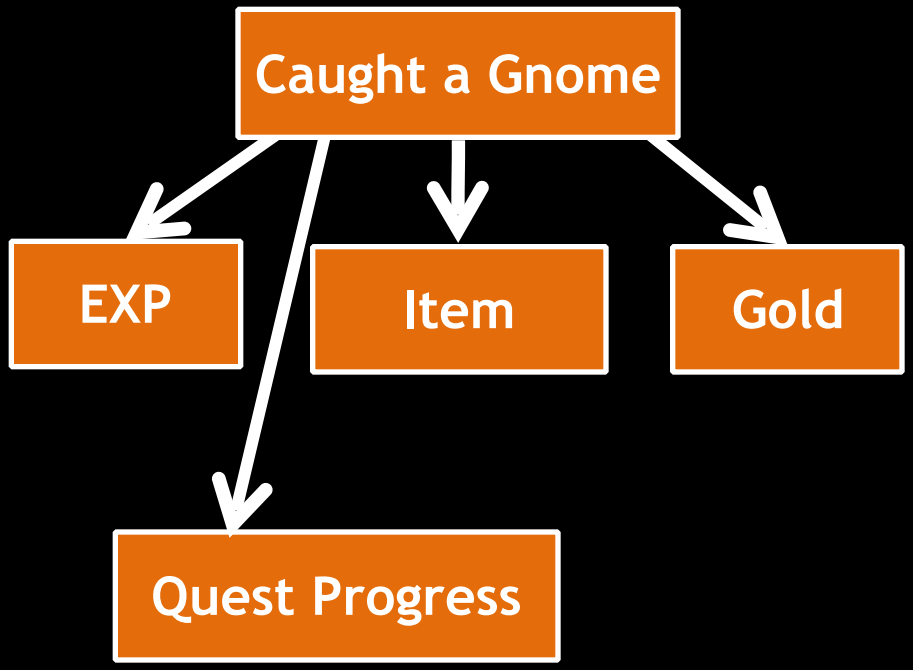
- non-blocking I/O
- no callbacks

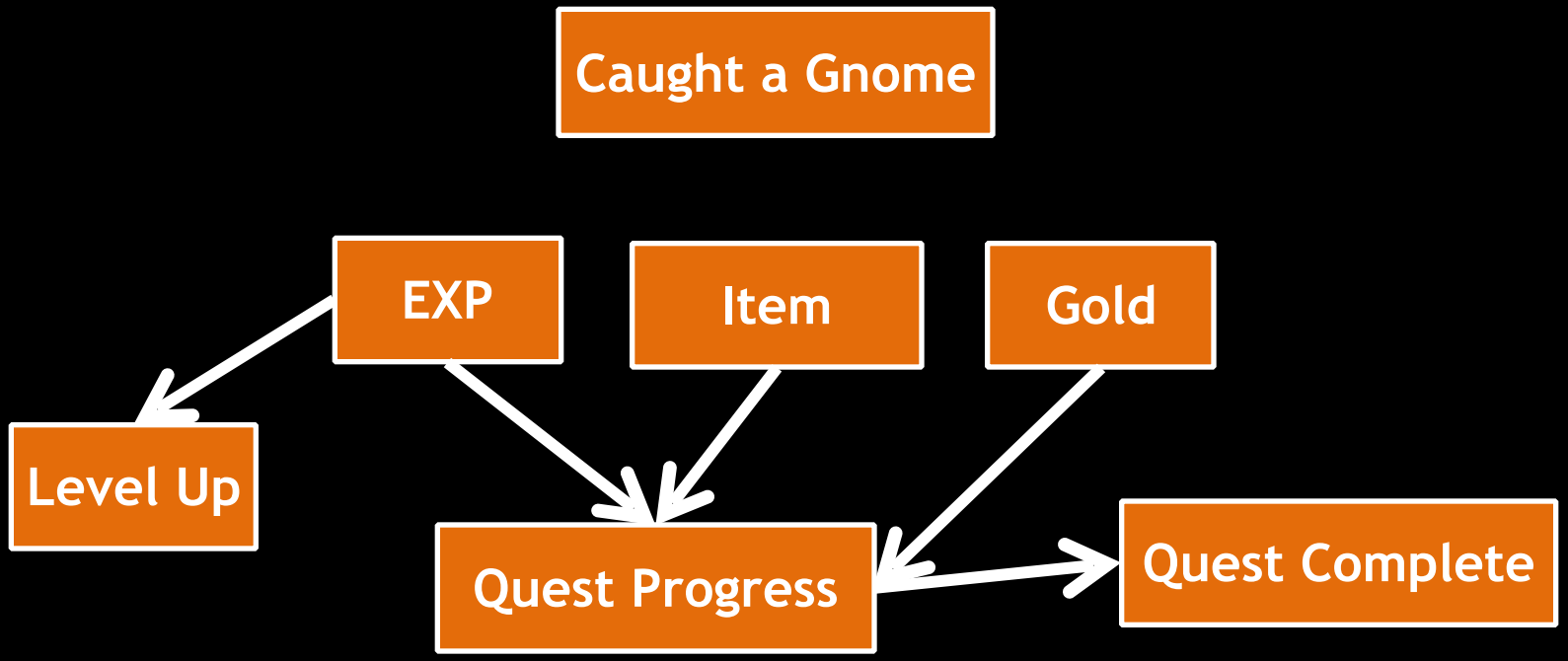
Pattern Matching

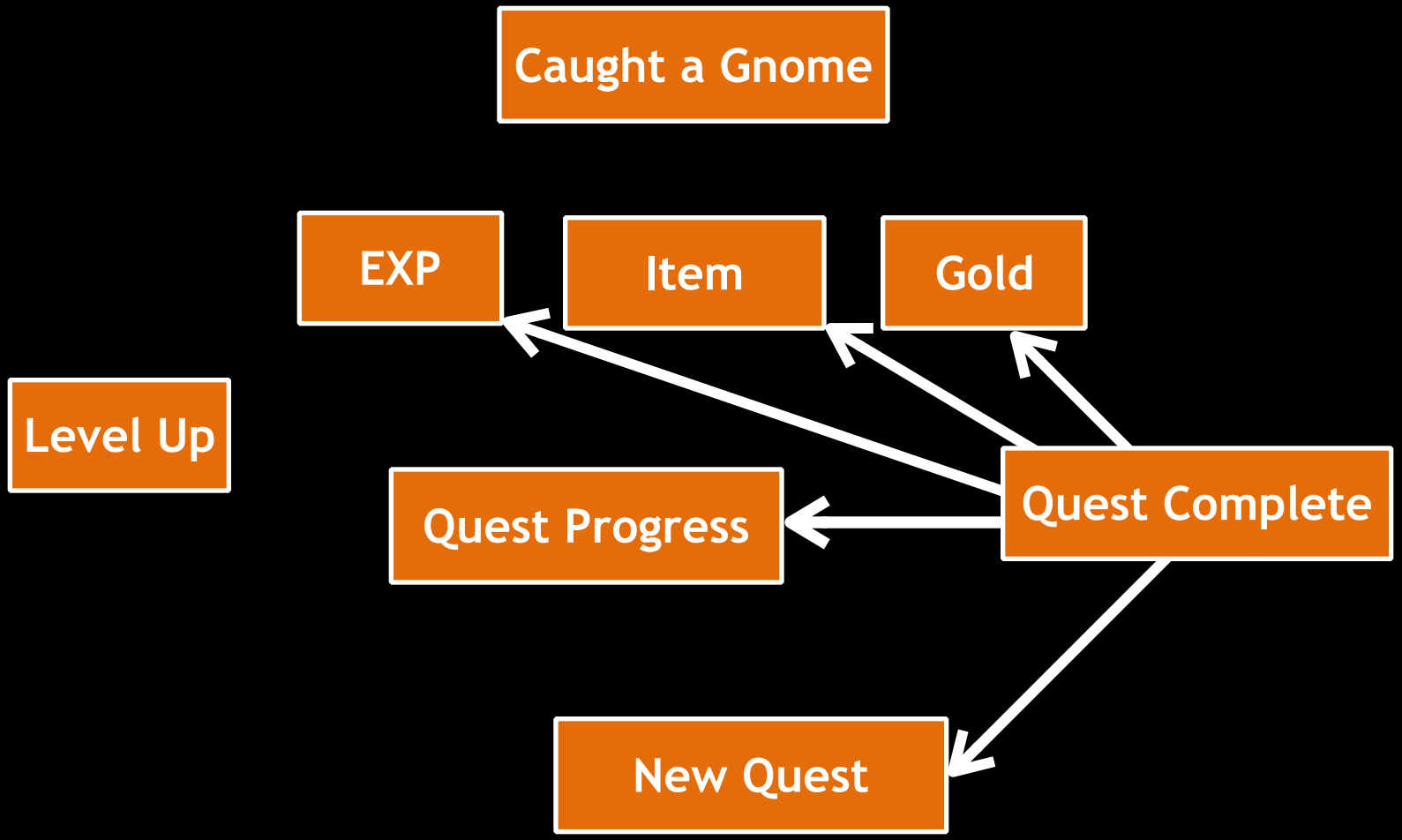
- clear & concise

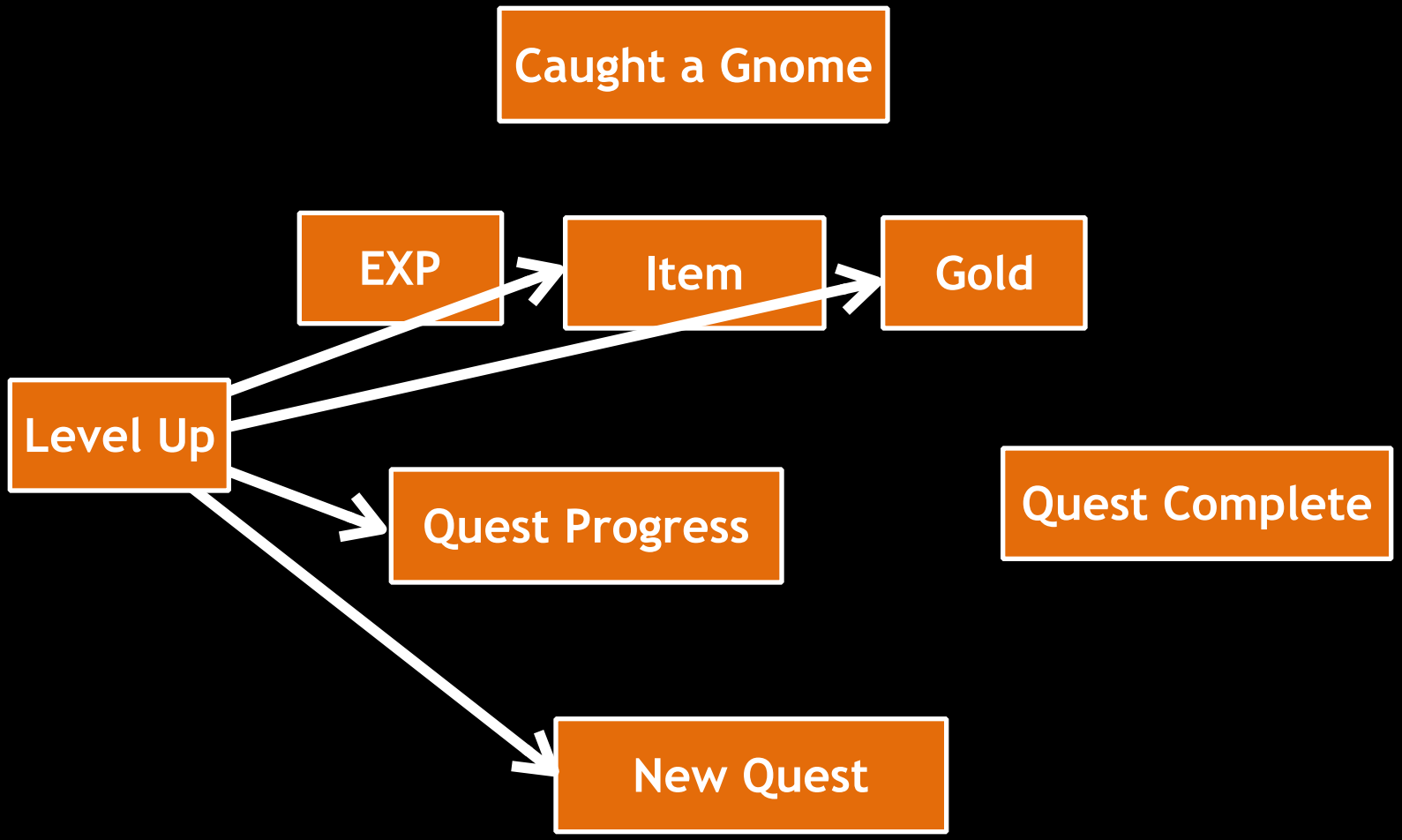

Thinking about quests
Broken from questing
Running towards quests
Eddie's Sketchbook

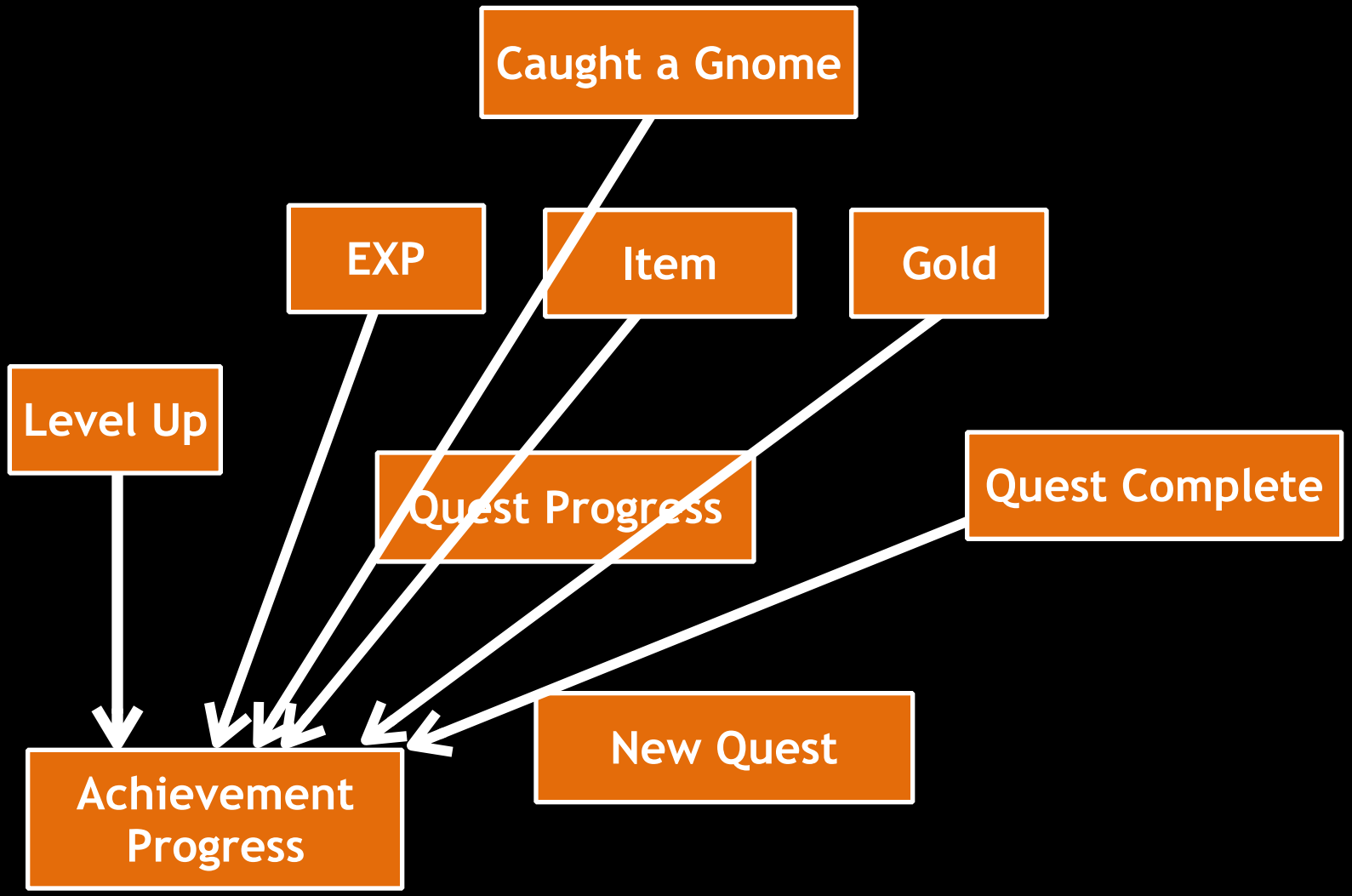
Caught a Gnome


Caught a Gnome
EXP
Item
Gold
Quest Progress

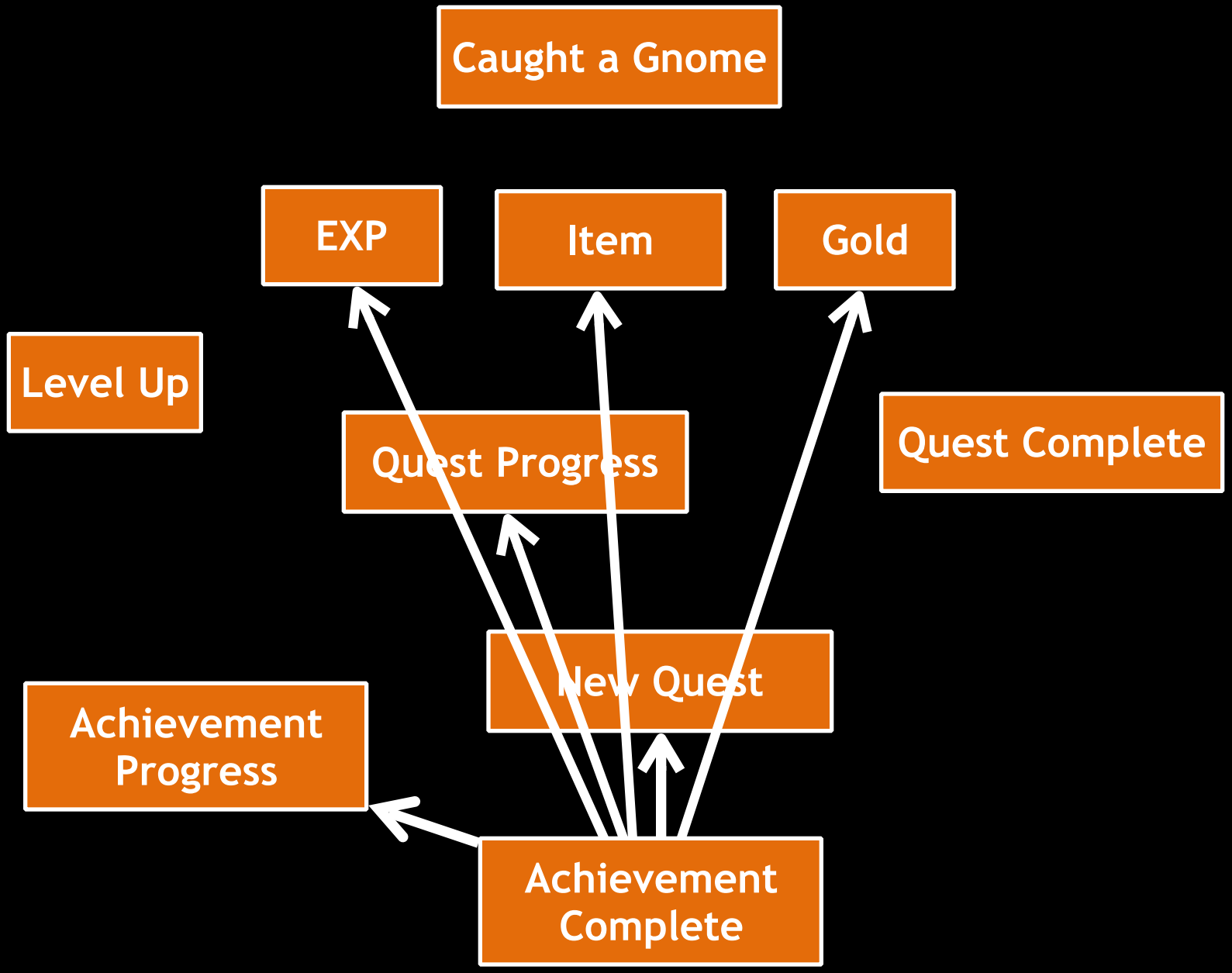

Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete


Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete
New Quest


Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete
New Quest


Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete
New Quest
Achievement Progress


Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete
New Quest
Achievement Progress
Achievement Complete

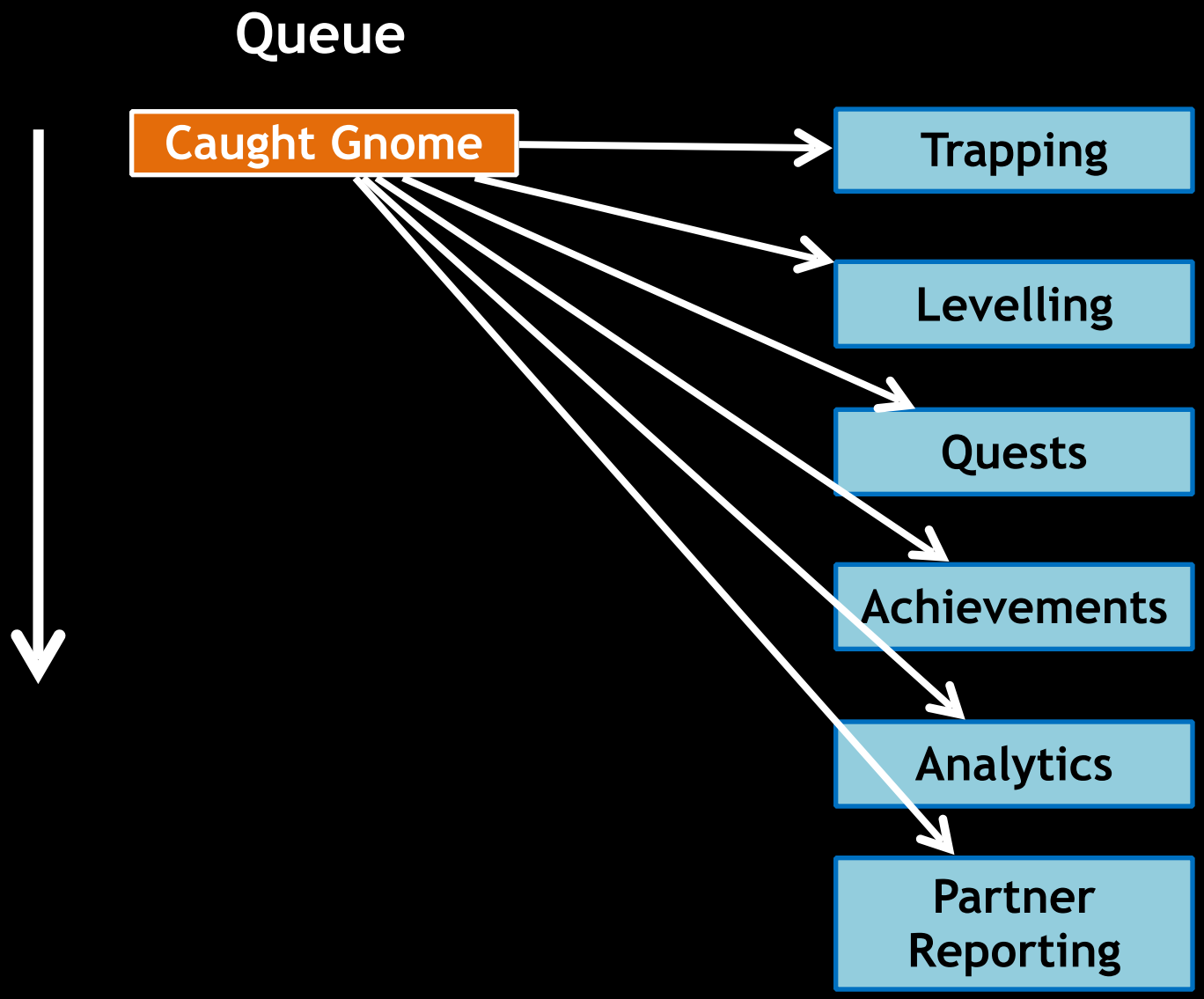

Caught a Gnome
EXP
Item
Gold
Level Up
Quest Progress
Quest Complete
New Quest
Achievement Progress
Achievement Complete

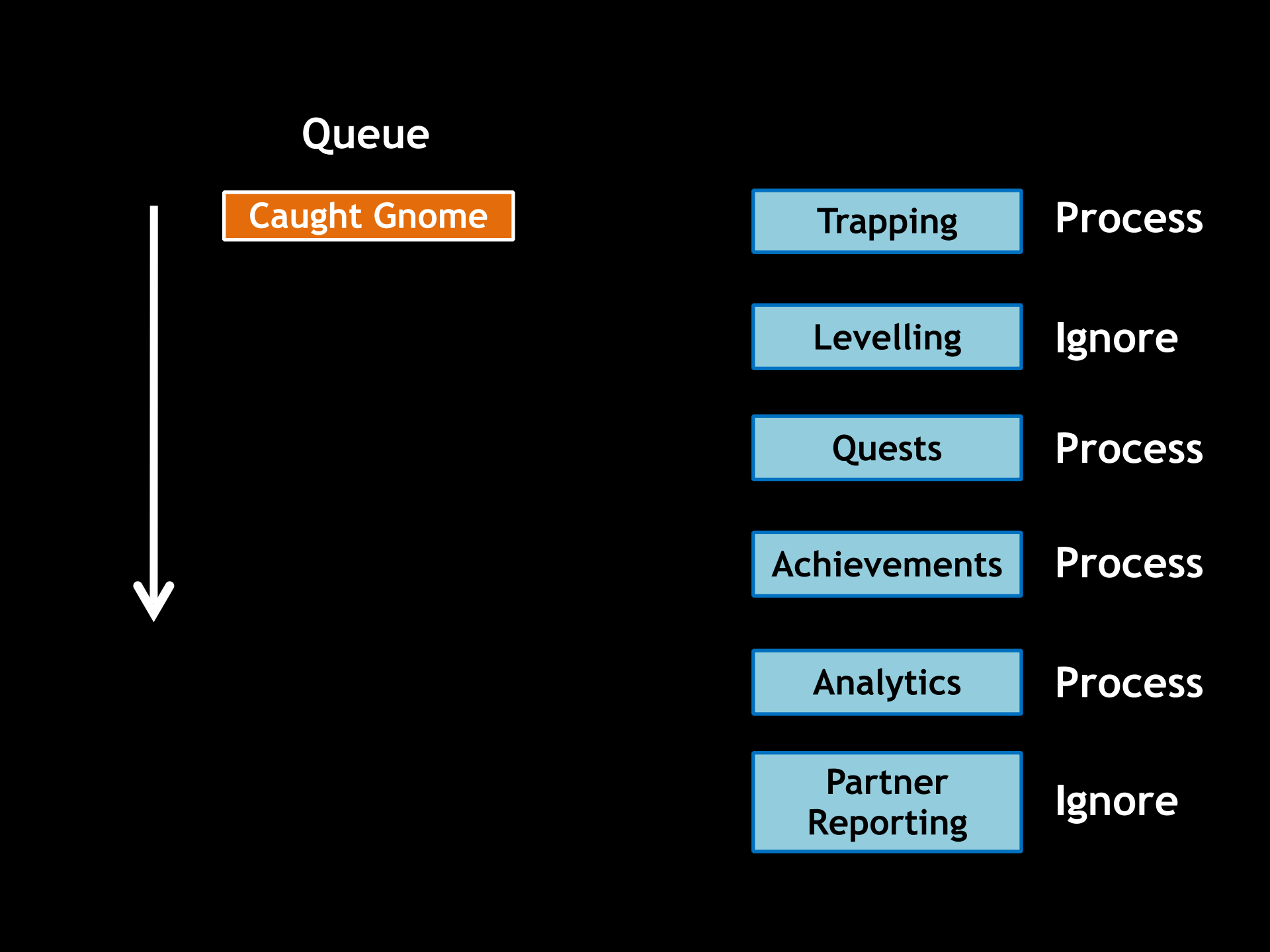
100+ actions

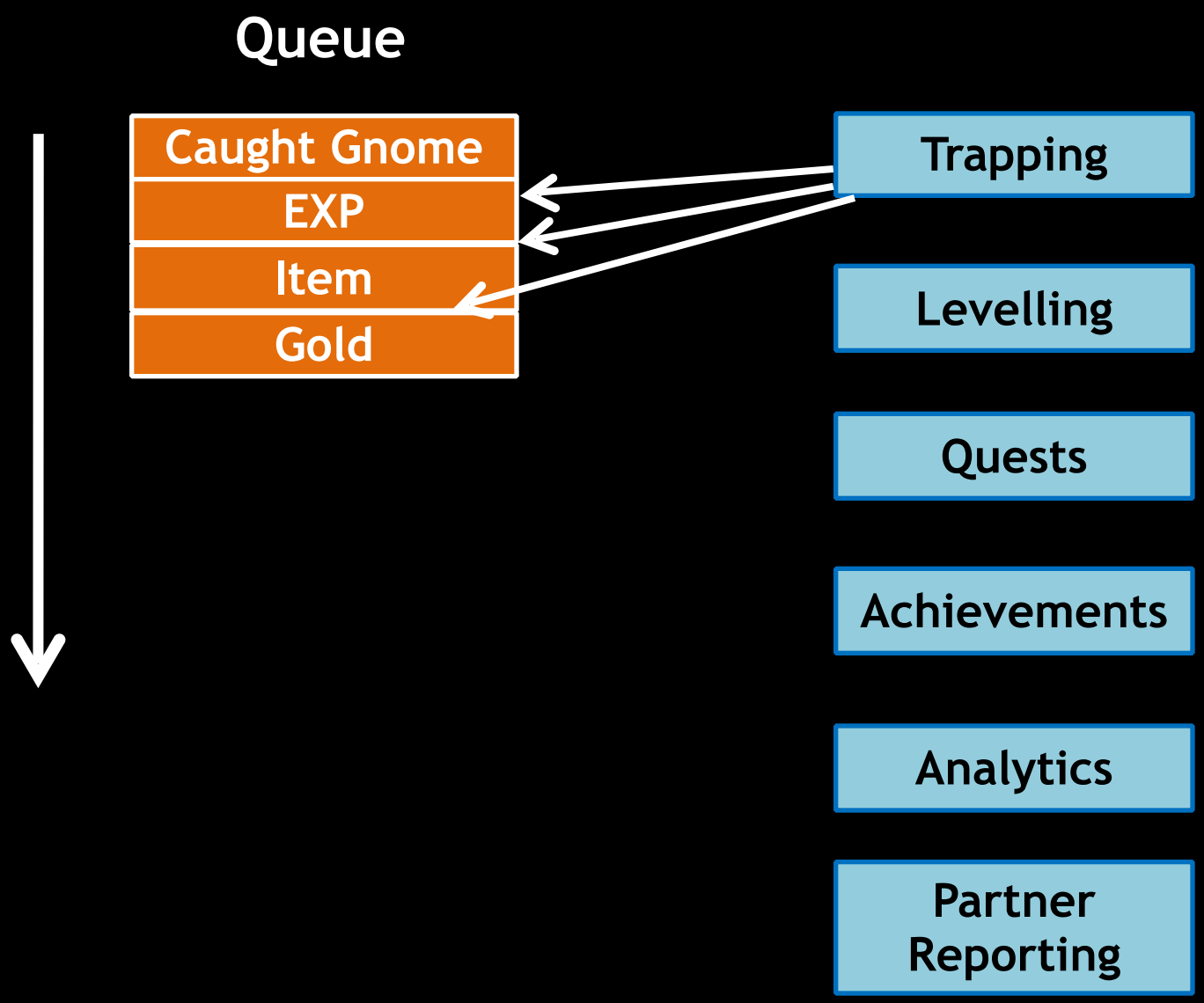
triggered by different abstraction layers

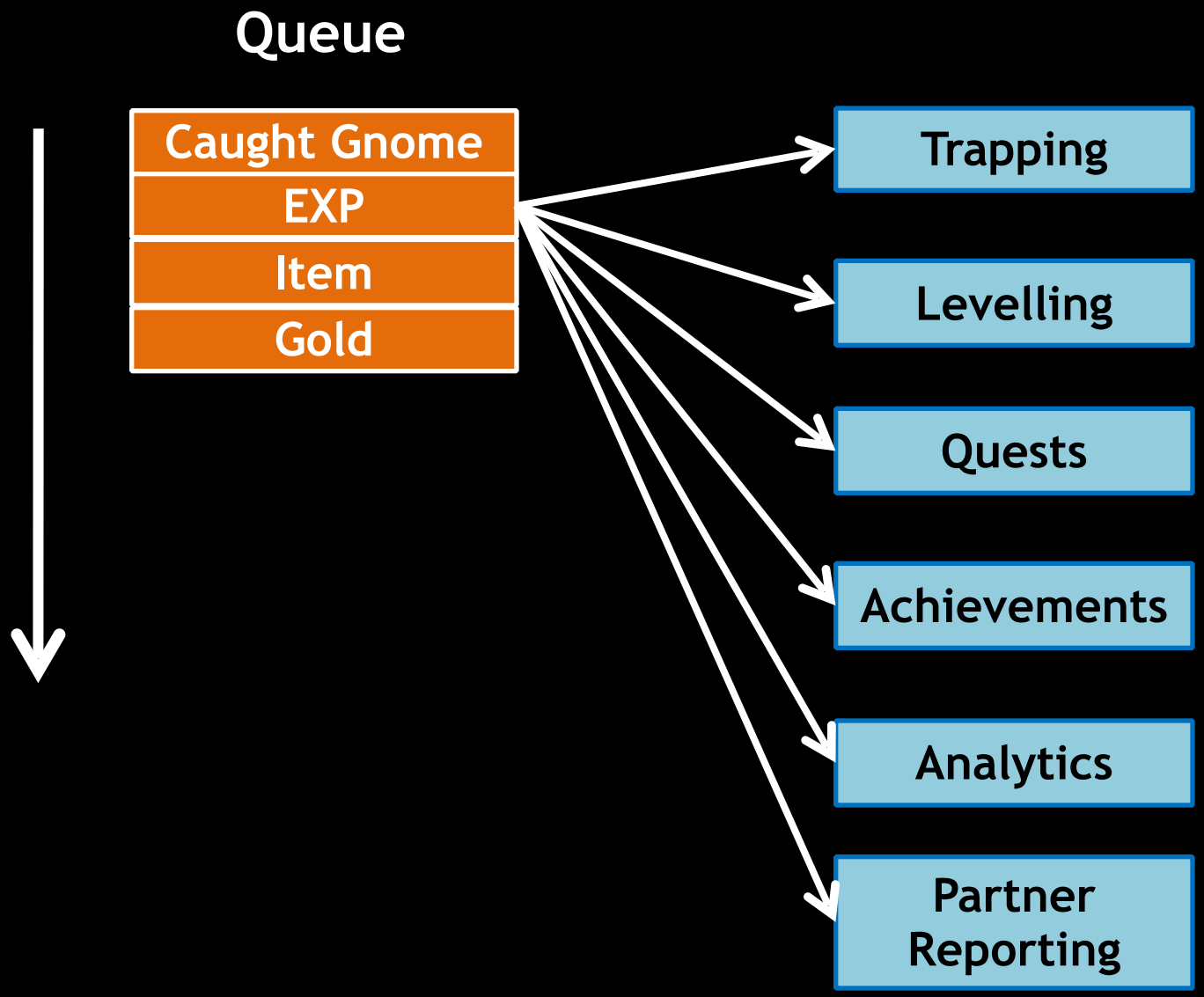
non-functional requirements

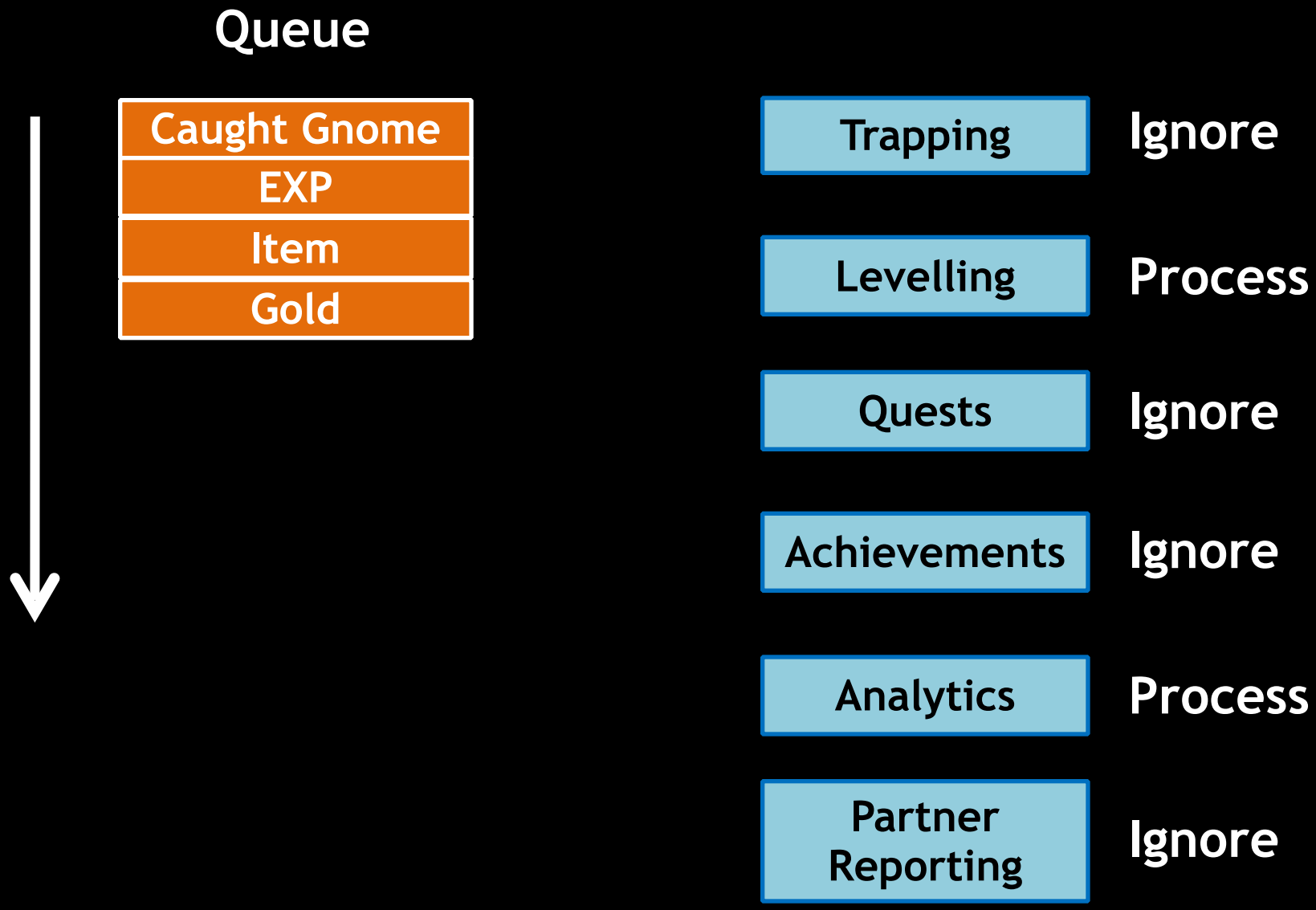
message-broker pattern

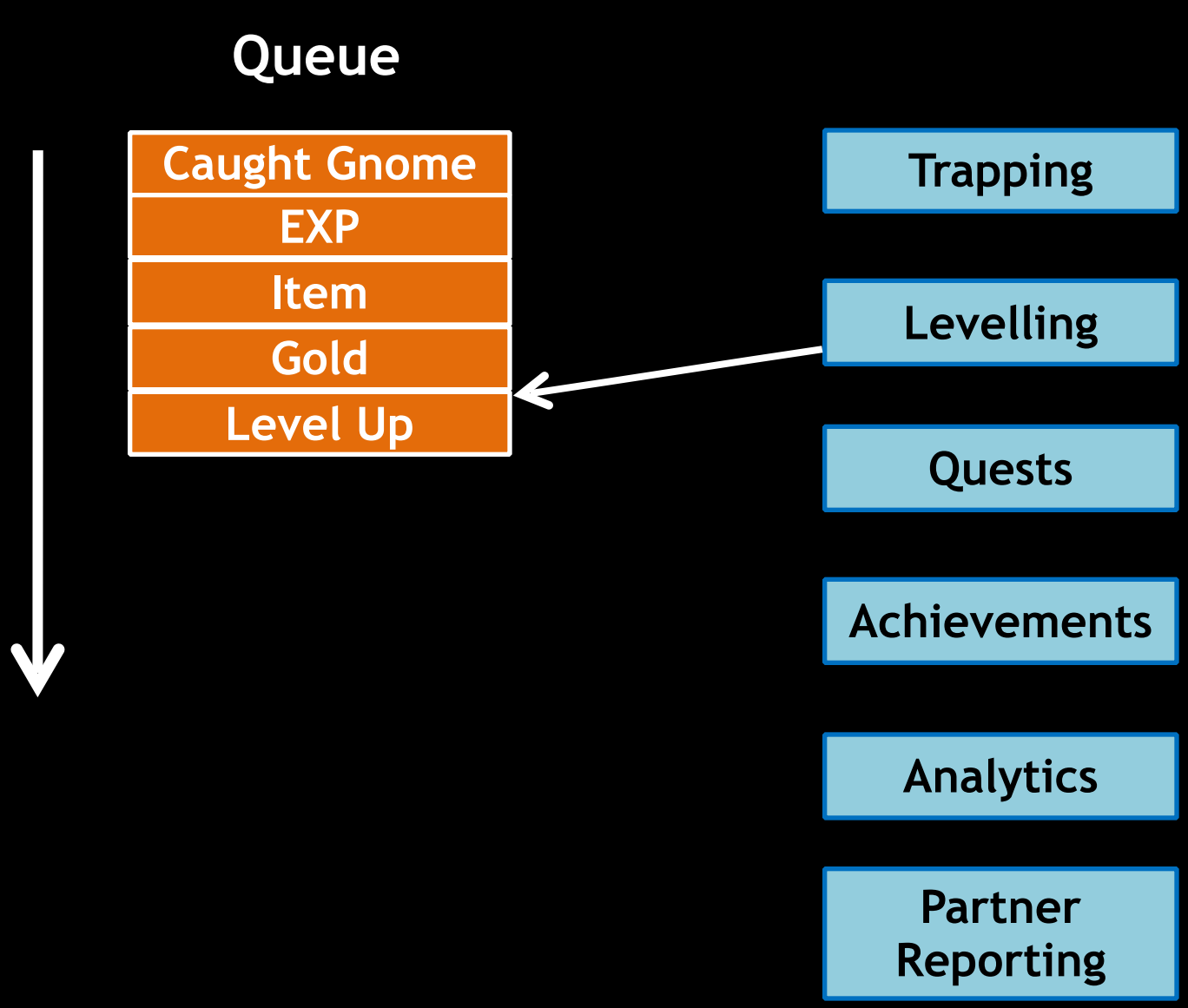

Queue
Caught Gnome
Trapping
Levelling
Quests
Achievements
Analytics
Partner Reporting

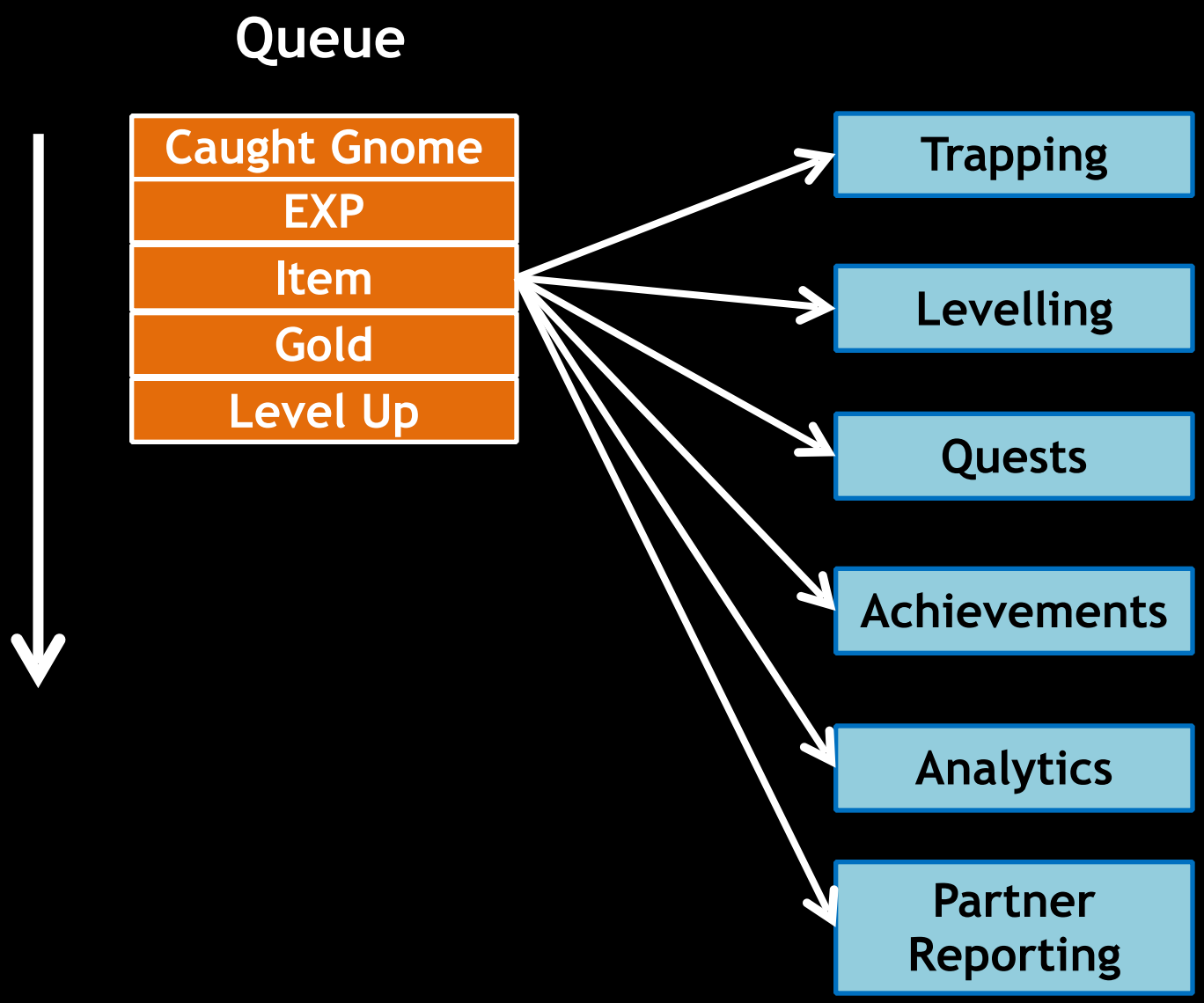

Queue
Caught Gnome
Trapping
Process
Levelling
Ignore
Quests
Process
Achievements
Process
Analytics
Process
Partner Reporting
Ignore


Queue
Caught Gnome
EXP
Item
Gold
Trapping
Levelling
Quests
Achievements
Analytics
Partner Reporting


Queue
Caught Gnome
EXP
Item
Gold
Trapping
Levelling
Quests
Achievements
Analytics
Partner Reporting


Queue
Caught Gnome
EXP
Item
Gold
Trapping
Ignore
Levelling
Process
Quests
Ignore
Achievements
Ignore
Analytics
Process
Partner Reporting
Ignore


Queue
Caught Gnome
EXP
Item
Gold
Level Up
Trapping
Levelling
Quests
Achievements
Analytics
Partner Reporting


Queue
Caught Gnome
EXP
Item
Gold
Level Up
Trapping
Levelling
Quests
Achievements
Analytics
Partner Reporting

need lots of facts

```
type Fact =
    | GotExp              of Exp
    | GotGold             of Gold
    | GotItem             of Item * Count
    | CaughtMonster       of Monster * Bait * Location
    | LevelUp             of OldLevel * NewLevel
    ...
```

```
type Reward =
    | GotExp            of Exp
    | GotGold           of Gold
    | GotItem           of Item * Count

type StateChange =
    | LevelUp           of OldLevel * NewLevel
    ...
```

```
type Fact =
    | StateChange       of StateChange
    | Reward            of Reward
    | Trapping          of Trapping
    | Fishing           of Fishing
    ...
```

```
let process fact =
    match fact with
    | StateChange(stateChange)  -> …
    | Reward(reward)            -> …
    | Trapping(trapping)        -> …
    …
```

C# interop

```
...
var fact = Fact.NewStateChange(
        StateChange.NewLevelUp(oldLvl, newLvl));
...
```

```
type IFact = interface end

type Reward =
    | GotExp                of Exp
    | GotGold               of Gold
    | GotItem               of Item * Count
    interface IFact
```

```
type IFact = interface end

type Reward =
    | GotExp                of Exp
    | GotGold               of Gold
    | GotItem               of Item * Count
    interface IFact
```

```
let process (fact : IFact) =
    match fact with
    | :? StateChange    as stateChange -> …
    | :? Reward         as reward      -> …
    | :? Trapping       as trapping    -> …
    …
    | _ -> raise <| NotSupportedFact fact
```

```
let process (fact : IFact) =
    match fact with
    | :? StateChange    as stateChange -> …
    | :? Reward         as reward      -> …
    | :? Trapping       as trapping    -> …
    …
    | _ -> raise <| NotSupportedFact fact
```

simple

flexible

extensible

saver

17th April
London

Leave your feedback on Joind.in!
https://joind.in/13652