GAMEPLAY PROGRAMMING IN A RICH AND IMMERSIVE OPEN WORLD

Digital Dragons 2016, May 16-17
Cracow, Poland

BARTŁOMIEJ WASZAK
GAMEPLAY PROGRAMMER AT UBISOFT QUÉBEC

Assassin's Creed Syndicate

- Historical action-adventure game with open-world gameplay
- Platforms: PS4, XBox One, PC
- Ubisoft Quebec as a leading studio + 9 other Ubisoft studios
- XIX-th century Victorian Era London
- Massive simulation of the vehicles: carriages, trains and boats
- Programming languages:
  - C++
  - Custom scripting language

Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- **Object model and world structure**
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

Game object model

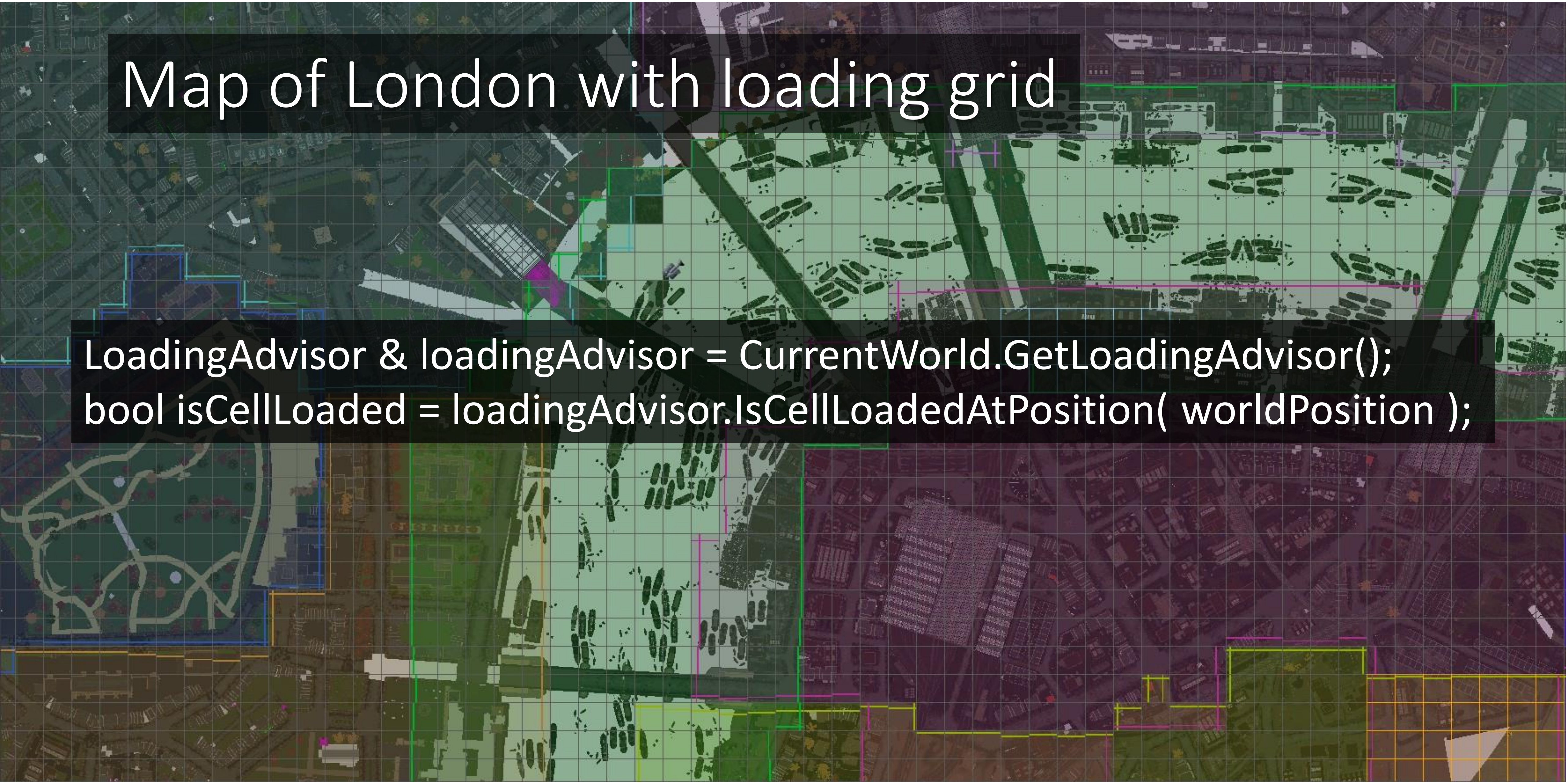
- **<u>Entity</u>**: main object represented in the world
- Entity: position + rotation, flags, components
- Components: configurable behaviours
- Example of entities: character, horse, street lamp, train's wagon, etc.
- Example of components:
  - Visual component
  - Rigid Body component
  - Behaviour (AI) component
  - Skeleton component

World structure

- World is a single object with added "world" components
- World components serve as managers for gameplay systems
- Examples of world components:
  - Railway System Component - trains
  - Carriage Manager
  - River System Component - boats

Map of London

Map of London with loading grid



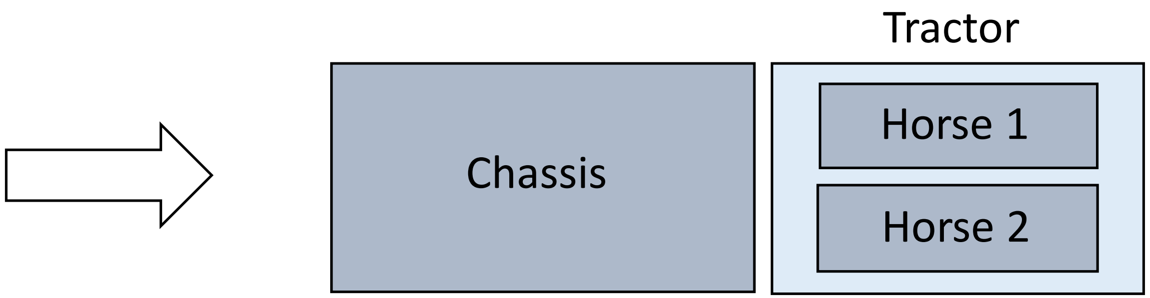
```
LoadingAdvisor & loadingAdvisor = CurrentWorld.GetLoadingAdvisor();
bool isCellLoaded = loadingAdvisor.IsCellLoadedAtPosition( worldPosition );
```

Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

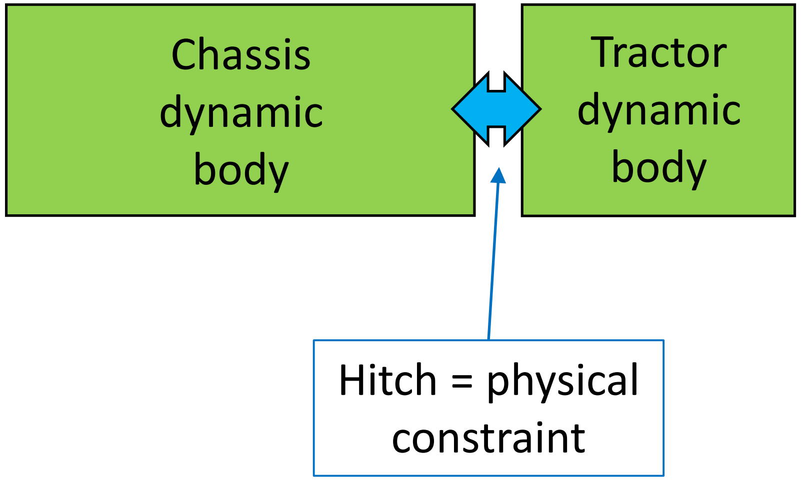
- Object model and world structure
- **Vehicles as part of the game world**
    - **Carriages**
    - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

Carriage entities



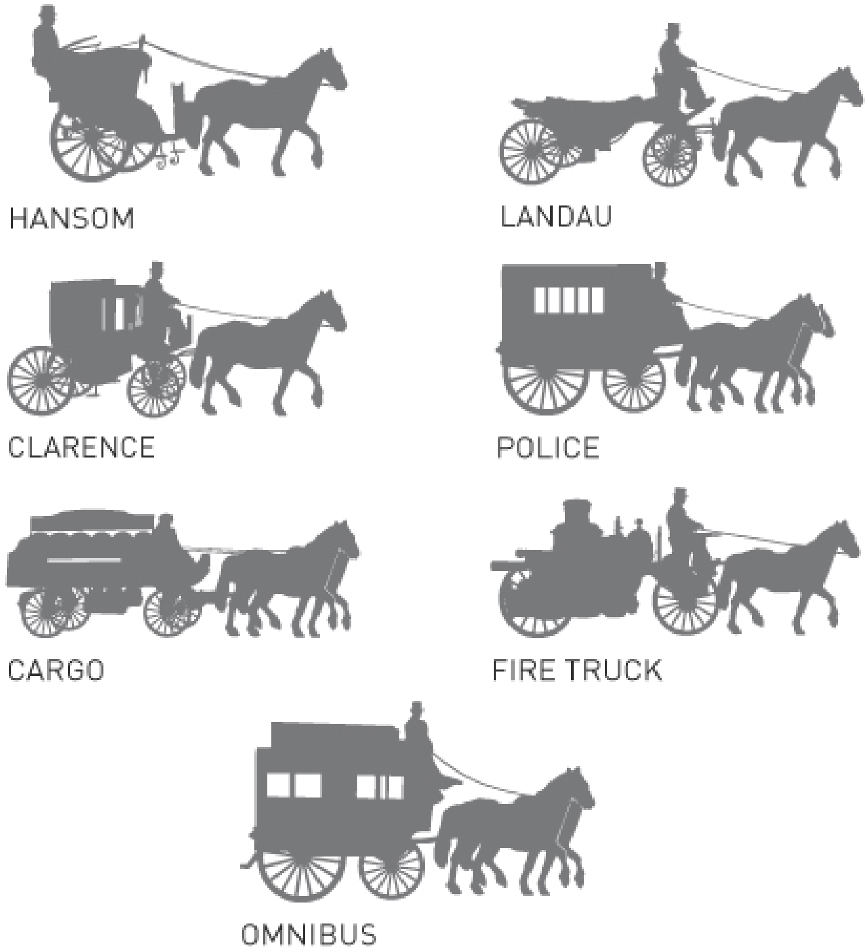
- Carriage is built from at least 3 entities:
- Carriage chassis (main body)
- Carriage tractor
- One entity representing each horse (1 or 2)

Carriage physics



- For physics engine, one dynamic rigid body is one entity
- Chassis – visuals + physics
- Tractor - no visual representation
- Horses - no physical simulation
- Hitch is modelled using constraints

Carriage-centric or entity-centric?



Existing code was entity-centric because character is one entity

More than one entity per carriage

A lot of redirections in the code

Traffic - streets of XIX-century London were full of carriages

Source: www.victorianlondon.org

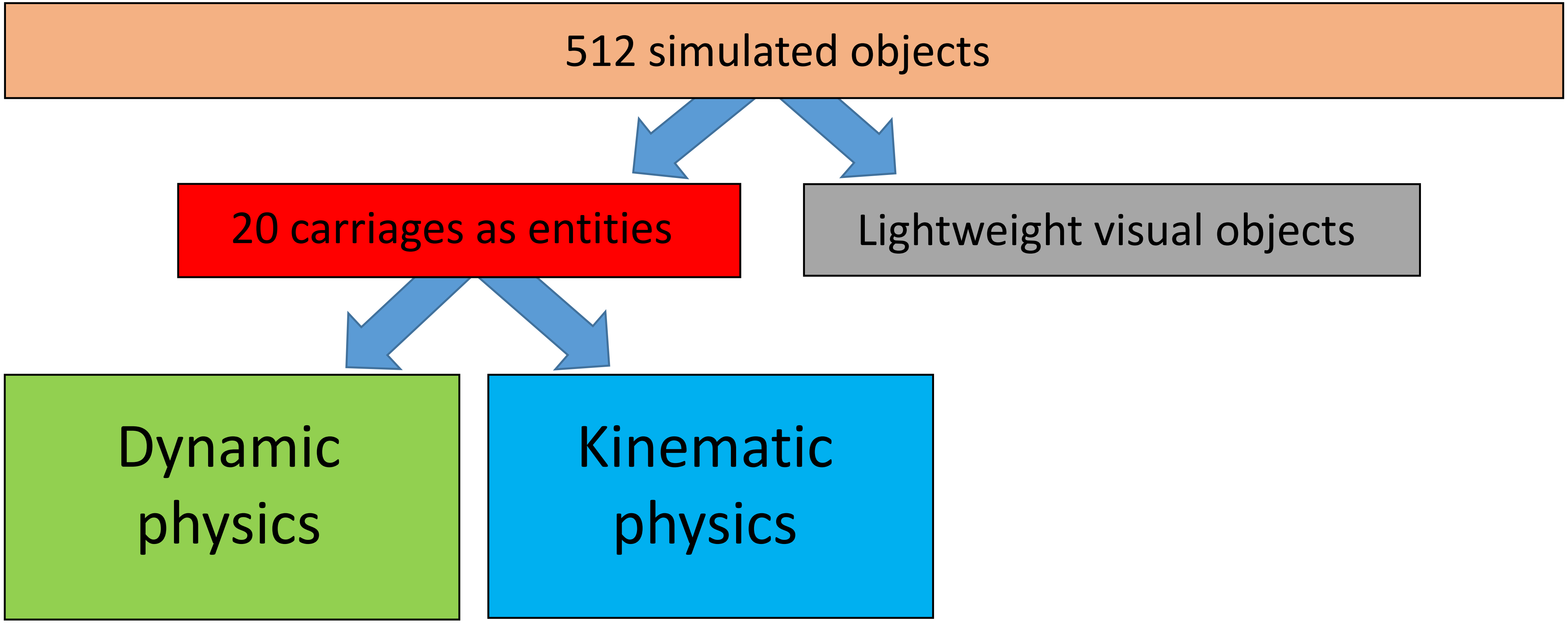
Source: Wikipedia

Traffic system in the game

Traffic - spawning

- Spawning the entity has relatively high time cost
- Instead of spawning and despawning we use a pool of reusable entities
- The pool is small - around 20 carriages
- Spawning is replaced with reinitialization for gameplay logic
- We try to perform only one reinitialization per frame
- Requests are processed based on the distance from camera and priority

Traffic - simulation



Presentation overview

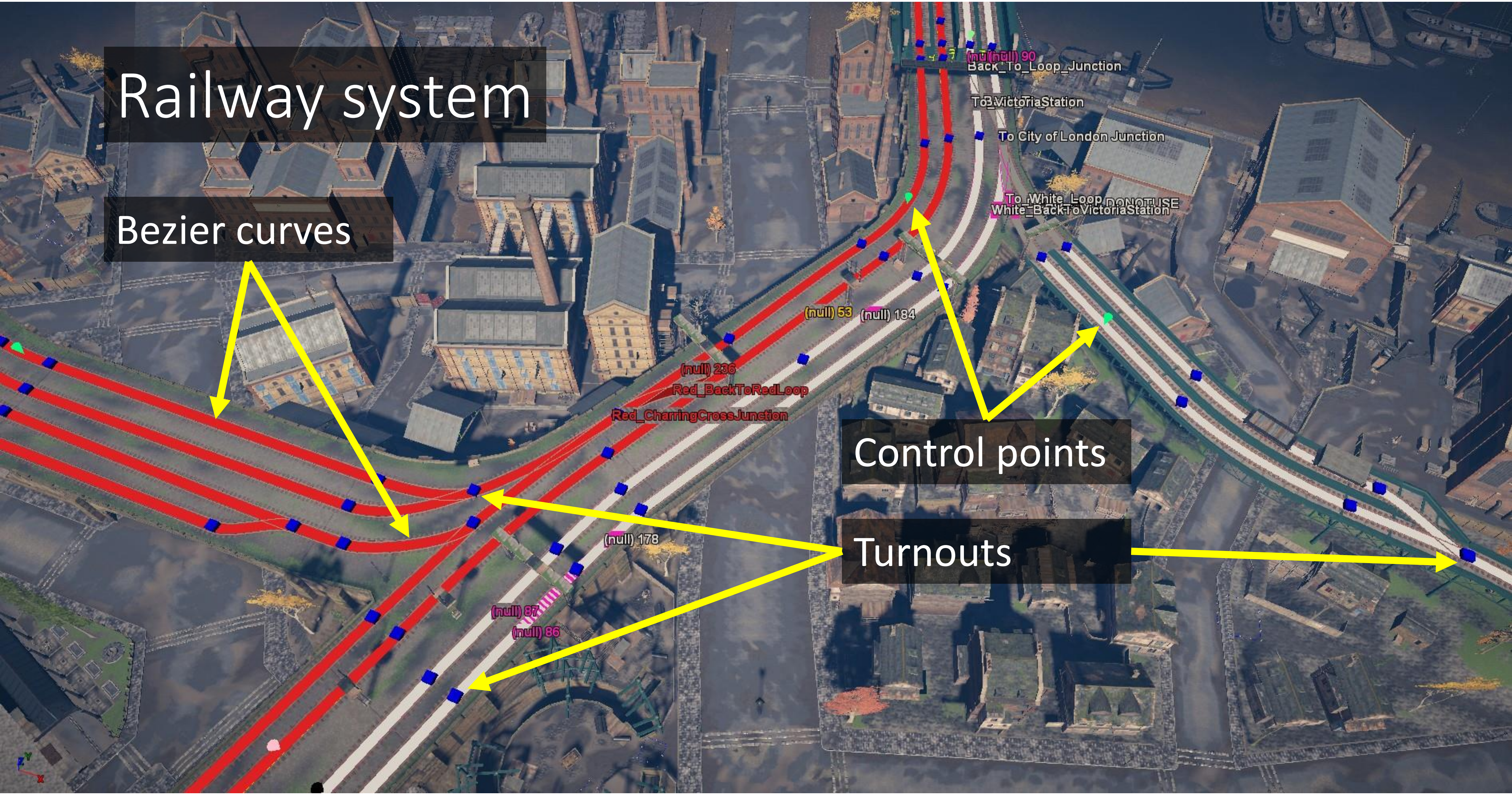
Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
    - Carriages
    - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

Trains in the game

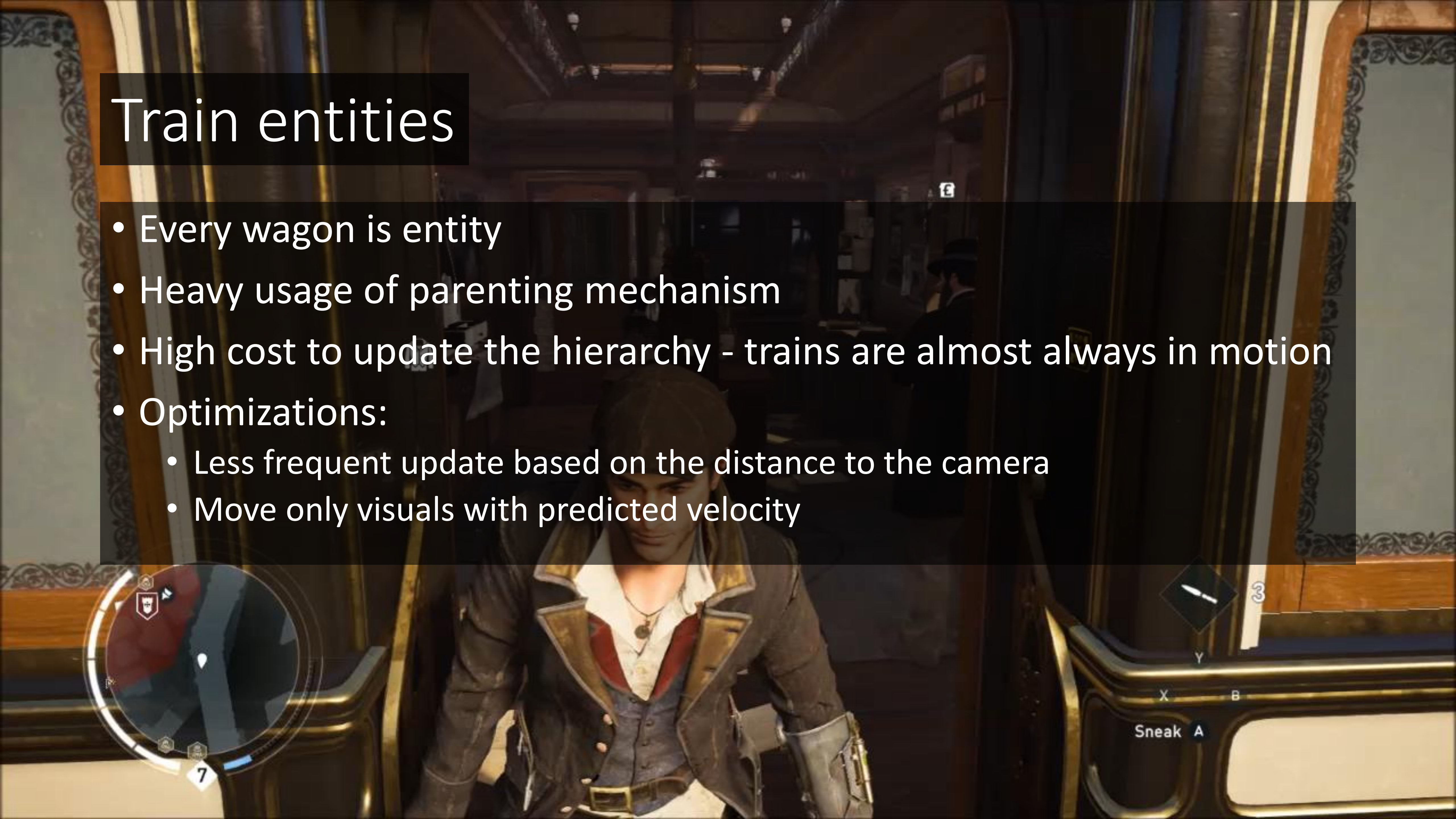
ASSASSIN'S CREED SYNDICATE

Railway system


Railway system
Bezier curves
Control points
Turnouts
Back_To_Loop_Junction
To City of London Junction
White_BackToVictoriaStation
(null) 53
(null) 184
(null) 236
Red_BackToRedLoop
Red_CharringCrossJunction
(null) 178
(null) 87
(null) 86

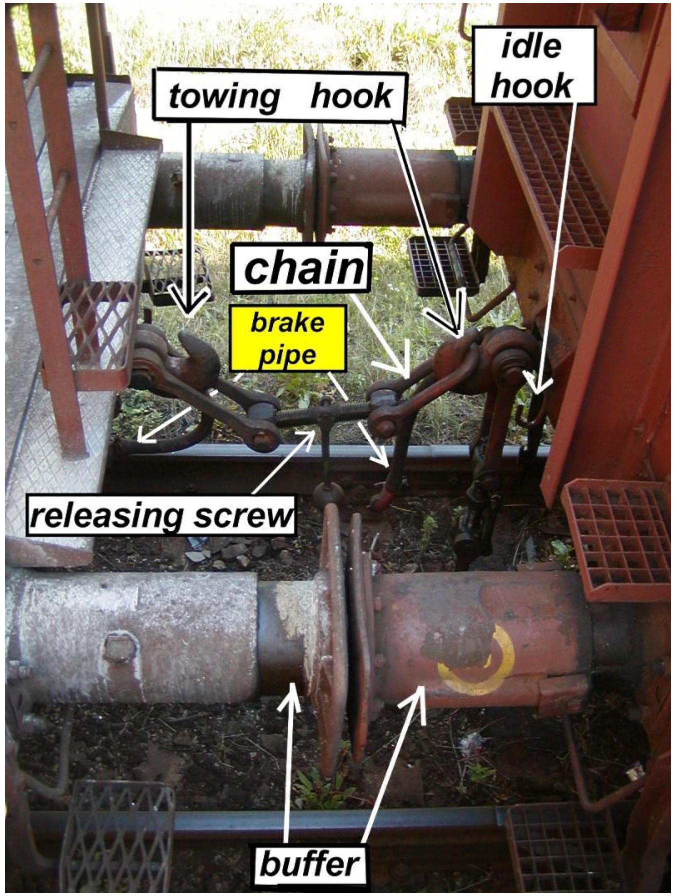
Train entities

- Every wagon is entity
- Heavy usage of parenting mechanism
- High cost to update the hierarchy - trains are almost always in motion
- Optimizations:
  - Less frequent update based on the distance to the camera
  - Move only visuals with predicted velocity



Trains - physics

- Custom physically-based simulator
- Movable coupling chain drives movement for all wagons
- Collision tests between wagons
- Custom constraints solver for coupling chain
- Momentum transfers during train's start-up and full stop



Source: Wikipedia

Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
    - Carriages
    - Trains
- **Data-driven approach**
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

Data-driven approach

ASSASSIN'S CREED SYNDICATE

- Designers control behaviours directly from the world editor
- Entities are created offline and loaded into the world
- Parameters can also be changed during the game's run-time

- Parameters represent directly values for member variables in C++:

Speed	11.3

→

```
float m_Speed;
```

→

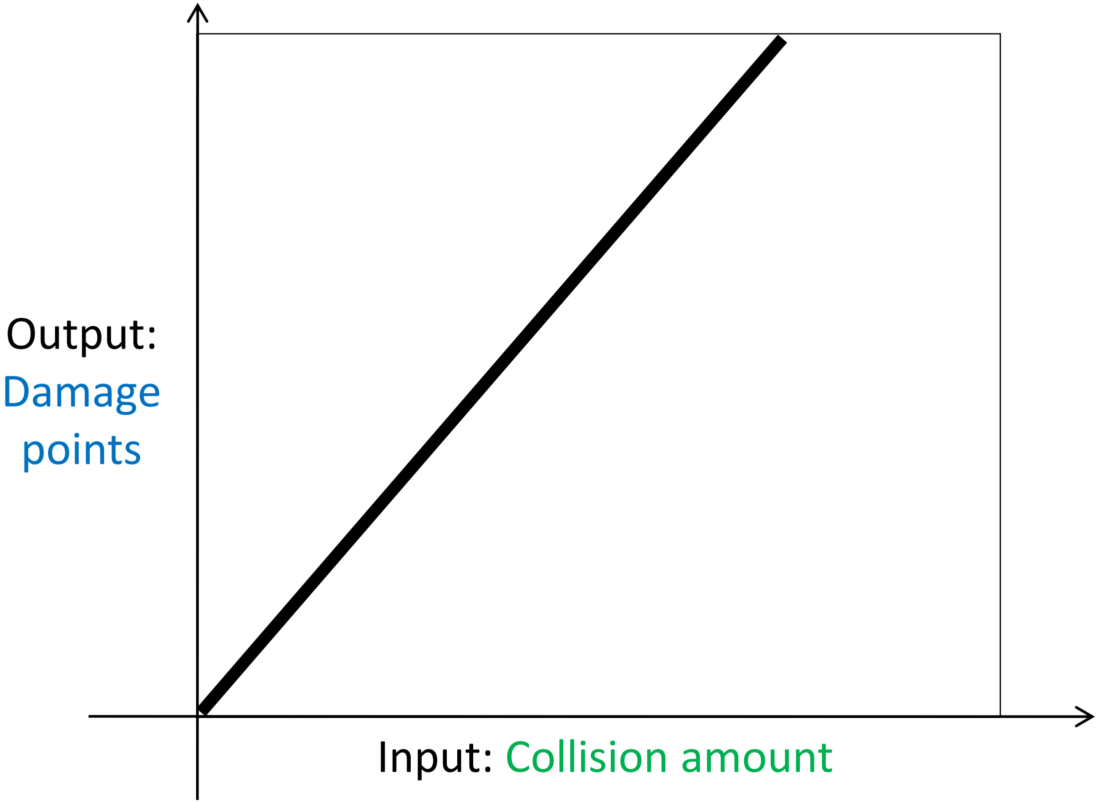
```
void SetSpeed(float NewSpeed);
float GetSpeed();
```

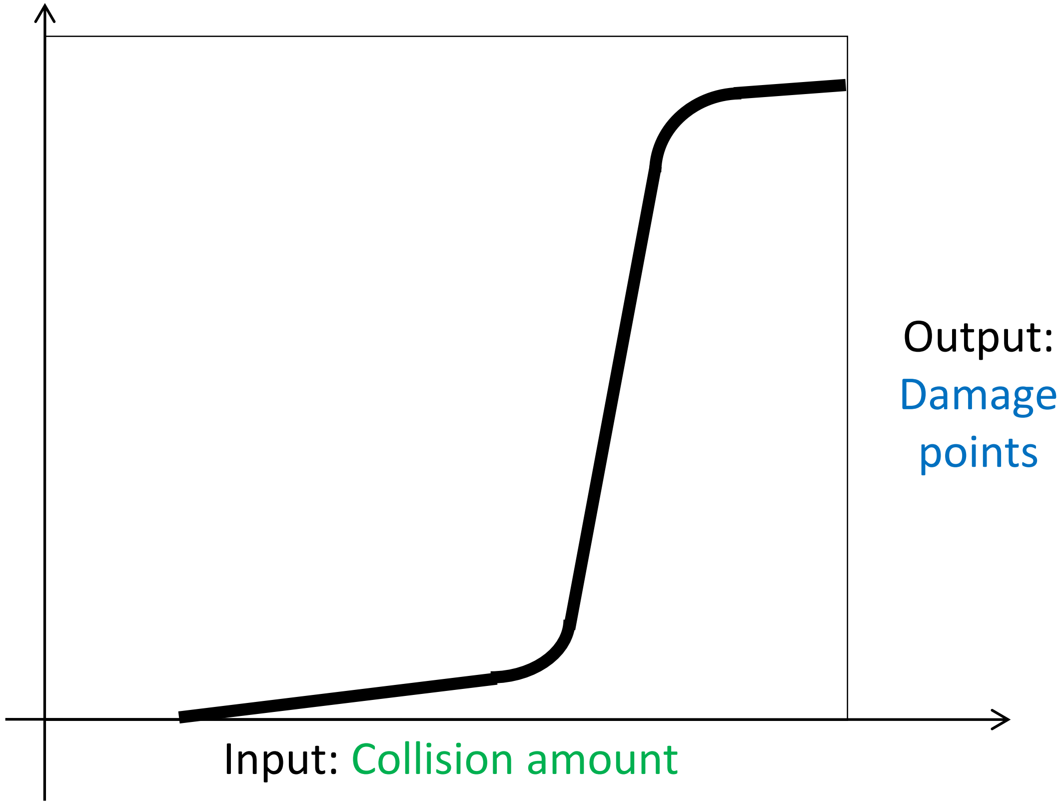
Data-driven approach

- To create advanced behaviours we need more than simple linear dependencies
- We used curves instead of linear coefficients
- We've used a lot of curves for carriage's handling parameters
- We used it also for such behaviours like hit points

Data-driven approach

- Simple linear dependency vs. more complex curve:





Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- **State machines**
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

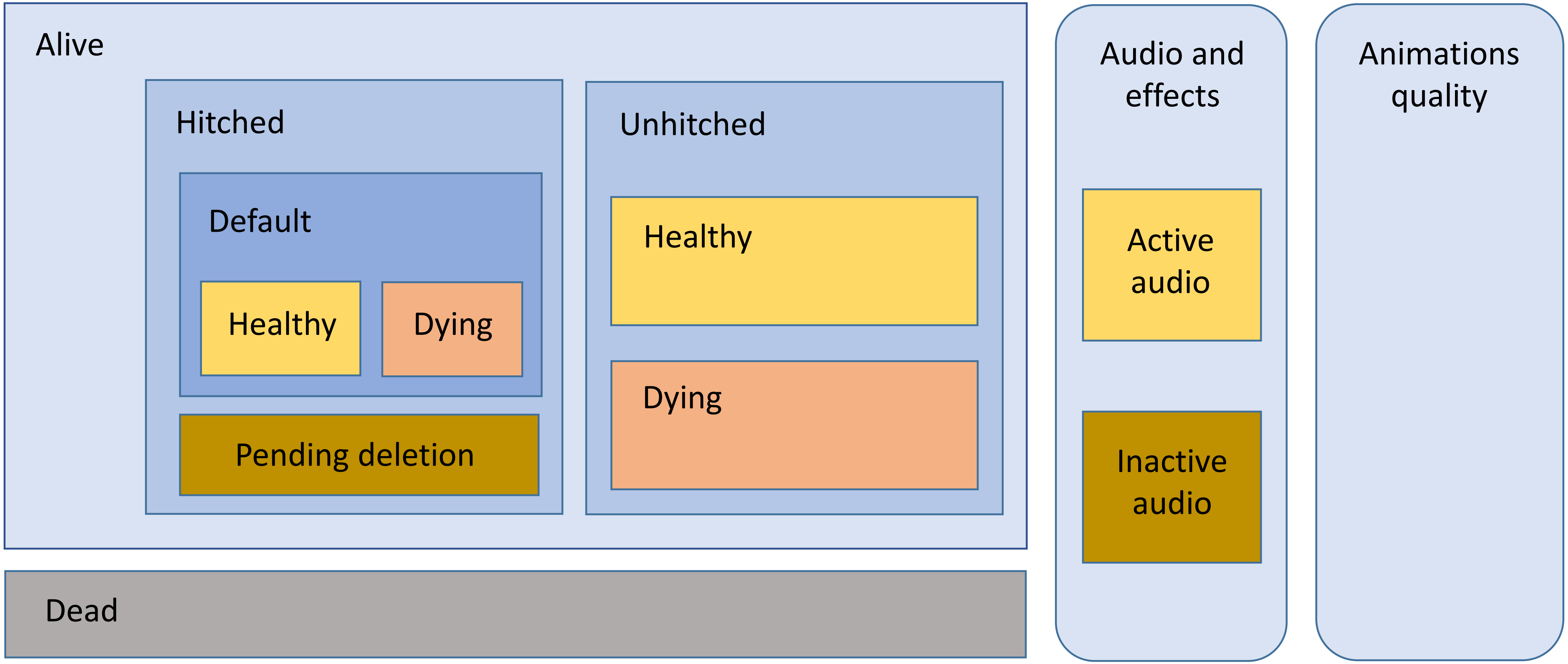
State machines

- Simple concept used throughout the gameplay code
- Basic building block for gameplay and AI logic
- Internal scripting language supporting directly state machines
- Examples of state machines:
  - Human state machine
  - Carriage state machine
  - Horse state machine
- Gameplay programming means working a lot with state machines

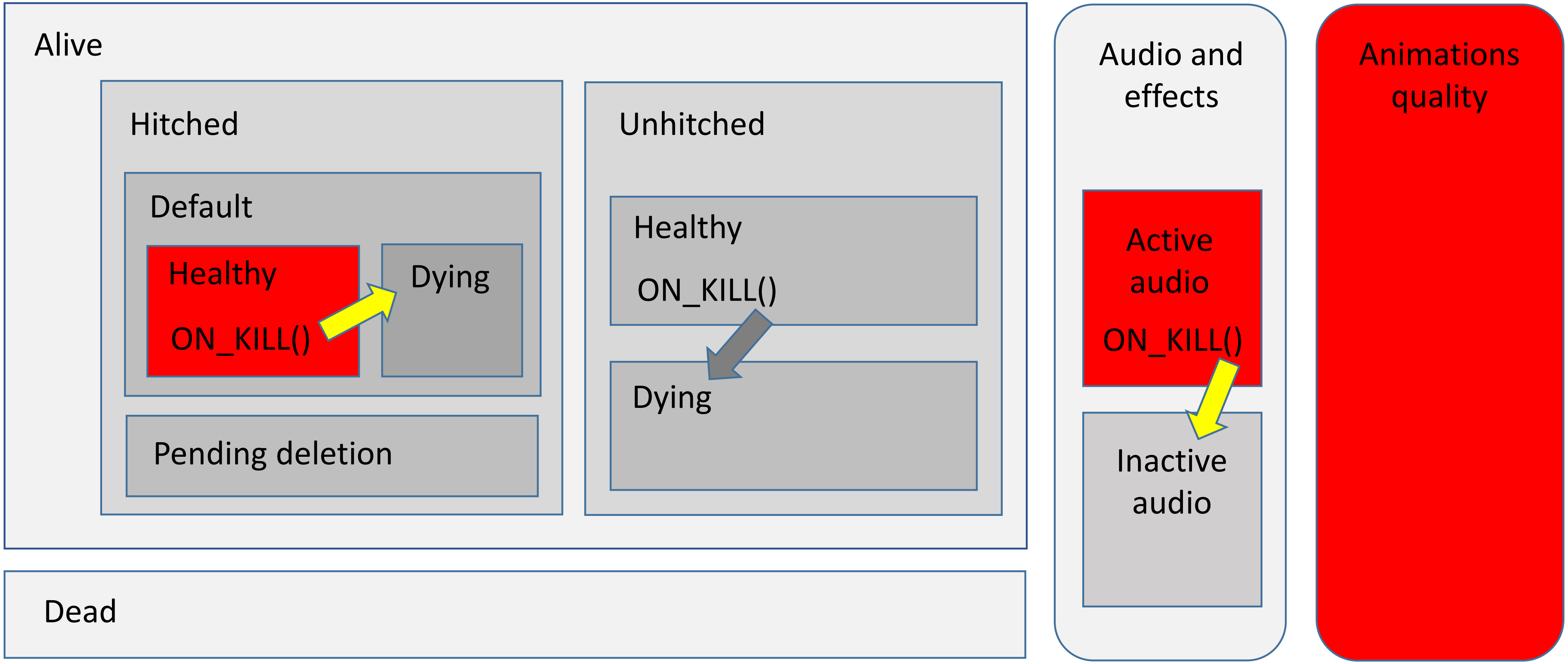
State machine structure

- States have constructor / destructor, enter / exit methods
- States can build hierarchies (substates)
- Exclusive states and additive states
- State machine is always in
  - one of the exclusive states
  - and maybe in one or more of additive states

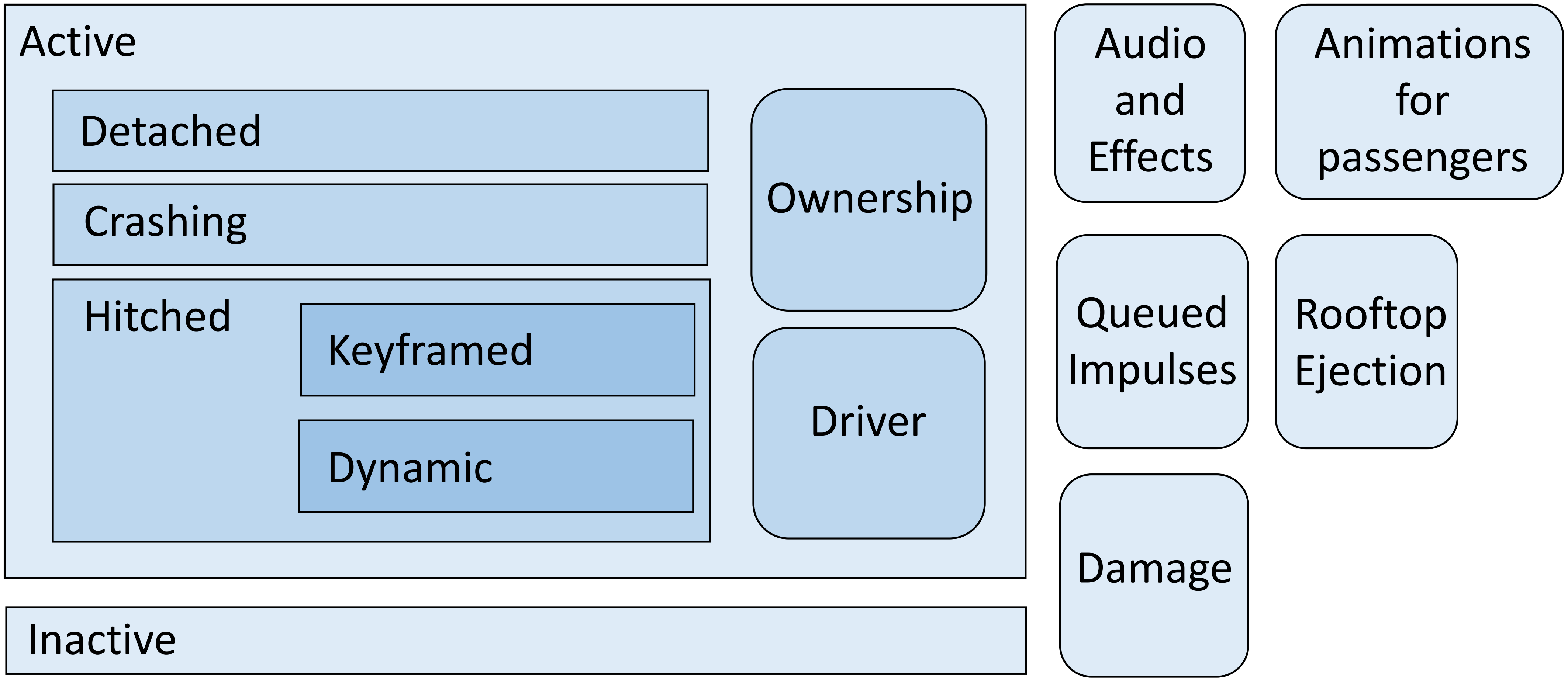
Horse state machine



Flow in horse state machine



Carriage state machine



Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- **Animation system and real-time control parameters**
- Event-driven gameplay logic
- Multithreaded environment
- Case study for the gameplay feature

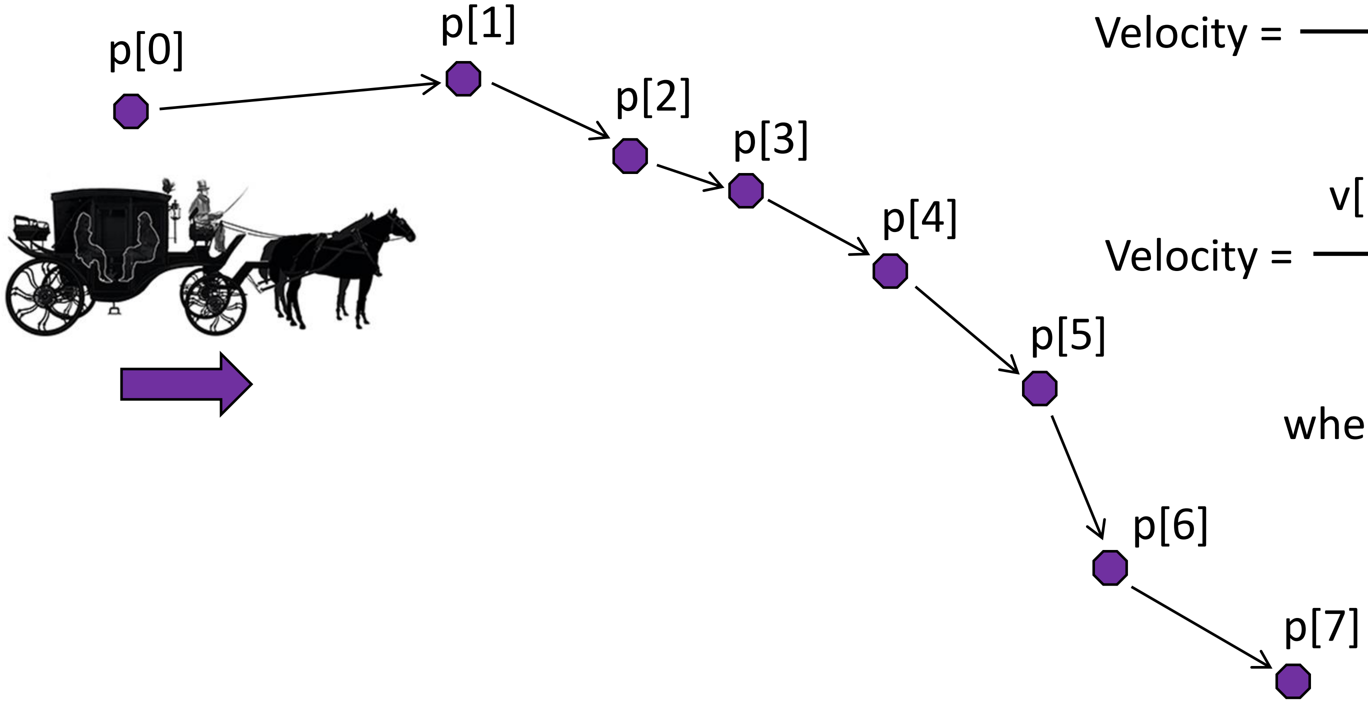
Real-time control parameters

- Data-driven control over the system through the set of simple parameters
- Parameters are using basic types like float, integer, enumeration, vector
- Examples:
  - Velocity, acceleration, emotional state (enumeration), impact direction
- Systems controlled through RTCPs: Animation, Audio and Special Effects
- Advantages:
  - Less coupled subsystems
  - Simple interfaces - easy to control
  - Easy to understand by non-programmers

Velocity as real-time control parameter

- Multiple behaviours for entities in the open-world game
- Different movement source: physics, animations, procedural
- Velocity - simple value, not so easy to calculate properly
- Solution:
  - External computation <u>independent</u> on movement source
  - Velocity calculation is based only on position changes
  - Will work even for the unknown future - any source of movement

Computation of velocity



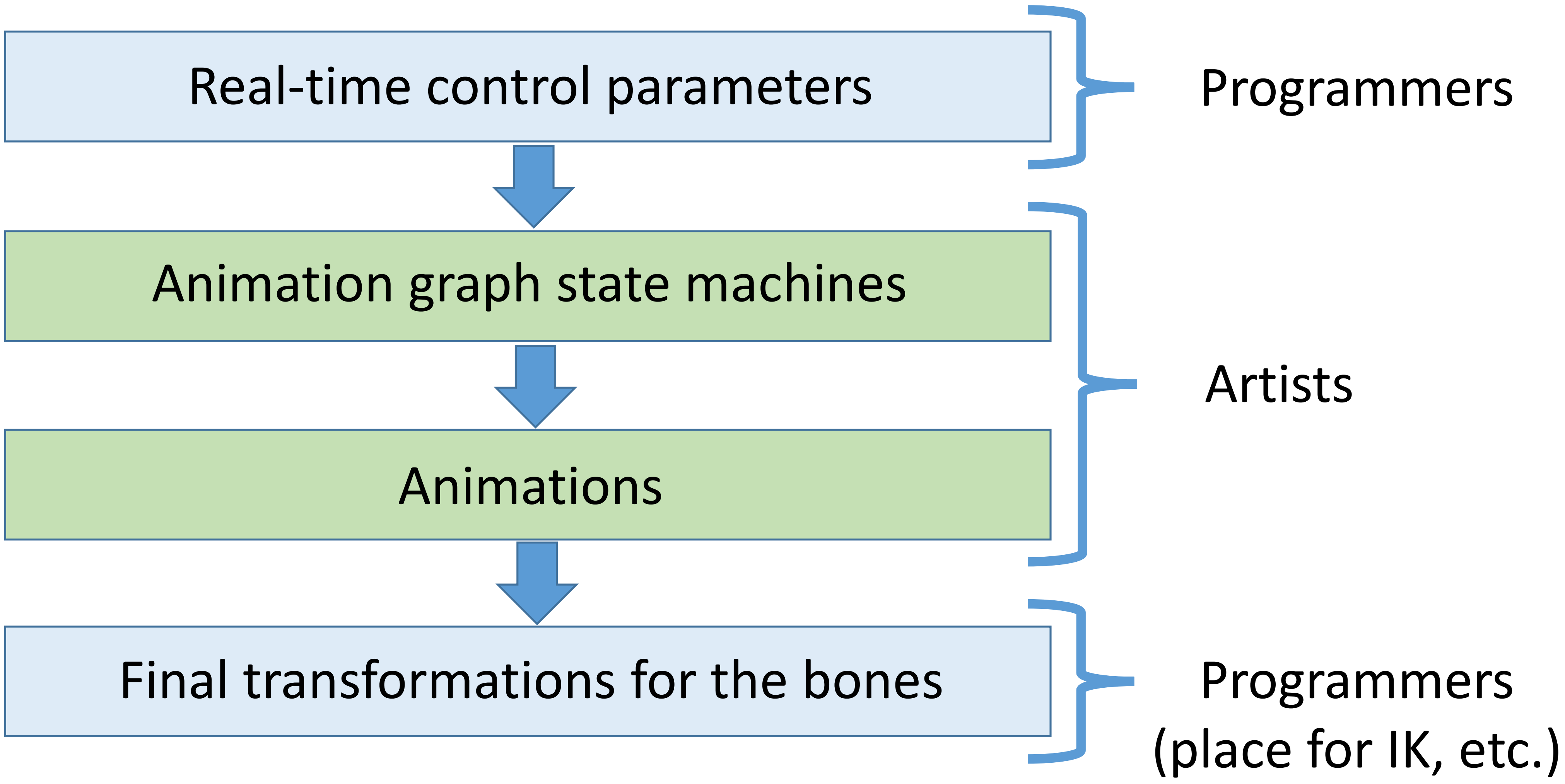
$$\text{Velocity} = \frac{\text{TotalDistance}}{\text{TotalTime}}$$

$$\text{Velocity} = \frac{v[1] + v[2] + v[3] + v[4] + v[5] + v[6] + v[7] + v[8]}{\text{TotalTime}}$$

where: $v[i] = p[i] - p[i-1]$

Real-time control parameters

Animation system - layers



Animation logic

- Very common pattern with the coupling between the state machine code and animation's RTCPs:

```
Alive_state
{
	enter_state
	{
		m_Entity->GetAnimComponent()->SetControlParameter( GeneralStates_Alive, eRTCP.GeneralState );
	}

	// ... specific code for Alive state

	exit_state
	{
		m_Entity->GetAnimComponent()->SetControlParameter( GeneralStates_Default, eRTCP.GeneralState );
	}
}
```

Horse animations

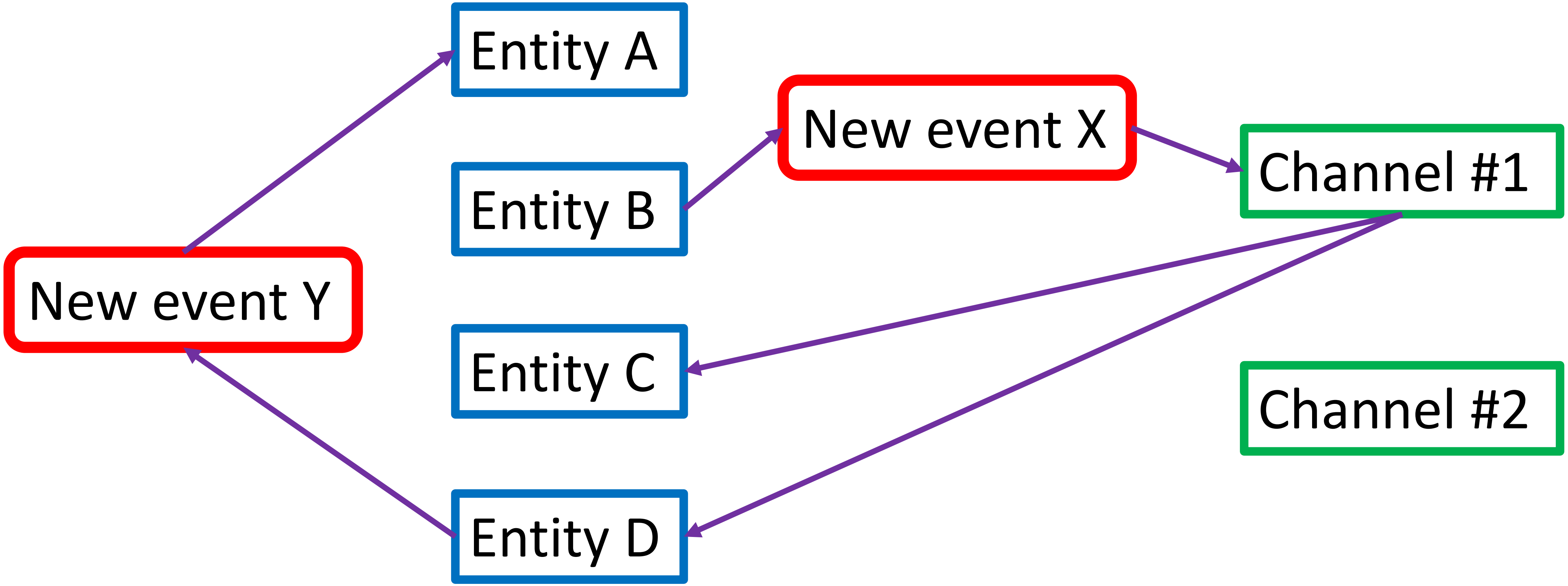
- Displacement: animation-driven vs. physics-driven
- The main character is animation-driven
- But horses are physics-driven
- Animation playback scaled with the current physical speed
- Different animations for speed ranges: walk, trot, gallop
- Hit reactions are additive animations

Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- **Event-driven gameplay logic**
- Multithreaded environment
- Case study for the gameplay feature

Event-driven gameplay logic



Event-driven gameplay logic

- Example of the event creation:

```
MyEventExplosion* newEvent = new MyEventExplosion;
newEvent->SetStrength( 4.0f );
SendEvent( channel, newEvent );
```

Receiving event from channel

- Method call upon receiving new event: OnExplosionEvent
- Can be called immediately or deferred
- Examples of events created in the game:
  - DestructibleHit
  - RammingEvent
  - CarriageTractorCollisionWithCharacter

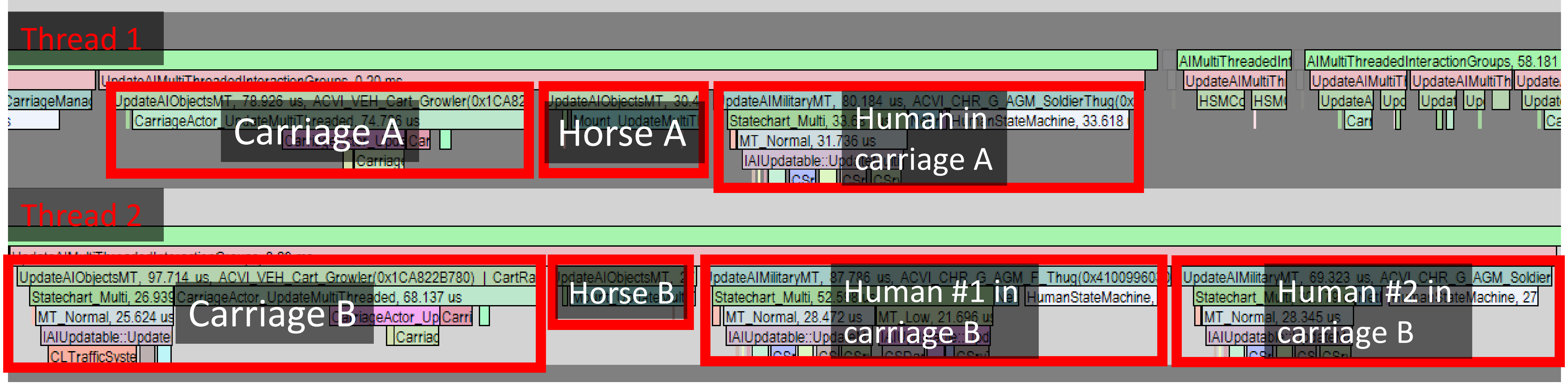
Presentation overview

Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- **Multithreaded environment**
- Case study for the gameplay feature

Multithreaded environment

- Parallel update for carriages, horses and humans



Presentation overview

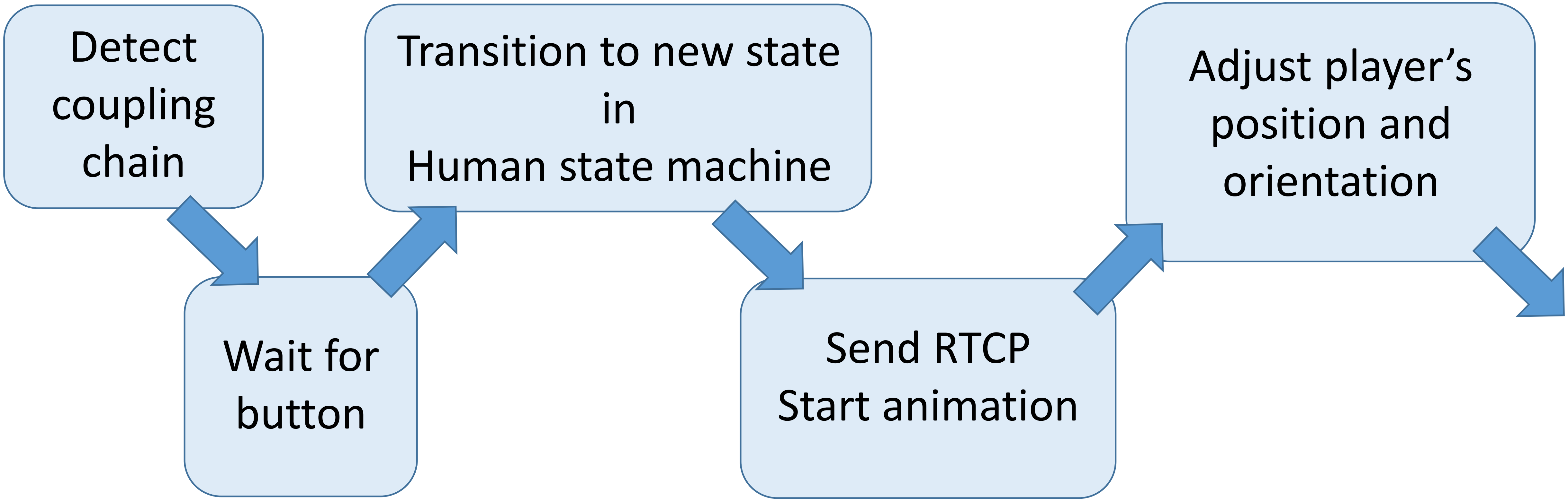
Gameplay programming with emphasis on new vehicles' systems:

- Object model and world structure
- Vehicles as part of the game world
  - Carriages
  - Trains
- Data-driven approach
- State machines
- Animation system and real-time control parameters
- Event-driven gameplay logic
- Multithreaded environment
- **Case study for the gameplay feature**

Decoupling train wagons

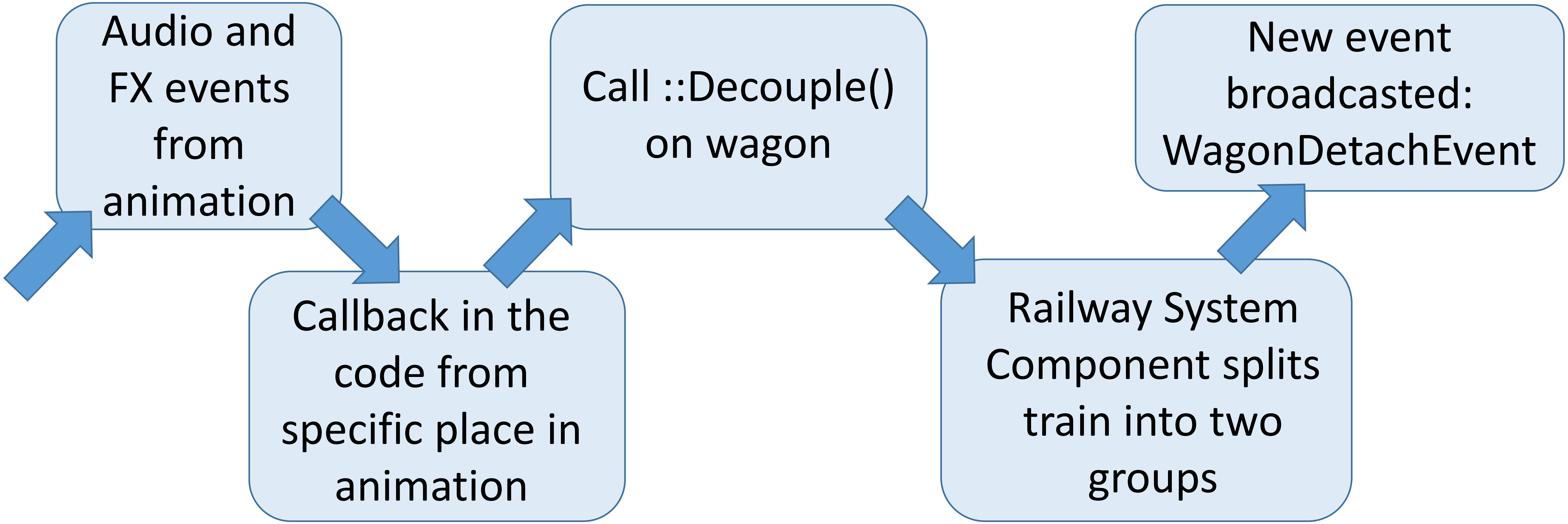
Decoupling train wagons

- What is happening in the gameplay code for this feature?



Decoupling train wagons

- What is happening in the gameplay code for this feature?



Thank You!

Questions?

bartlomiej.waszak@ubisoft.com

http://bwaszak.com

Thanks to:

Pierre Fortin, Dave Tremblay, James Carnahan, Matthieu Pierrot, Maxime Begin, Vincent Martineau, Martin Bedard, Marc Parenteau and Leszek Godlewski.