Game Architecture

Entity Systems

Today’s Agenda

- What is an entity?
- Traditional models.
- Entity as collection of components.
- Entities as a database.

What is an entity?

What is an entity?

An entity is something that exists as itself, as a subject or as an object, actually or potentially, concretely or abstractly, physically or not.

What is an entity?

An entity is something that exists as itself, as a subject or as an object, actually or potentially, concretely or abstractly, physically or not.

- Thanks, Wikipedia.

What is an entity?

An entity is defined by what it can do.

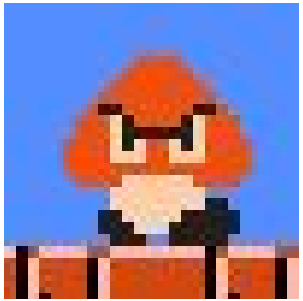
What can an entity do?

What can an entity do?

- Animate

What can an entity do?

- Animate
- Follow a path



What can an entity do?

- Animate
- Follow a path
- Attack

What can an entity do?

- Animate
- Follow a path
- Attack
- Explode

What can an entity do?

- Animate
- Follow a path
- Attack
- Explode
- Become visible (or not)

What can an entity do?

- Animate
- Follow a path
- Attack
- Explode
- Become visible (or not)
- Be selected

What can an entity do?

- Animate
- Follow a path
- Attack
- Explode
- Become visible (or not)
- Be selected
- Get picked up

What can an entity do?

- Animate
- Follow a path
- Attack
- Explode
- Become visible (or not)
- Be selected
- Get picked up
- Make sounds


SCORE 1520
LIVES

An assumption going forward

- Entities can do lots of different things.
- We are going to have lots of them.
- Not all of them will do the same things.

What is the goal?

- We want some kind of abstraction to simplify interactions among our many varied entities and higher level systems..
- And we don’t want to pay too much for that abstraction.

What is the goal?

- **We want some kind of abstraction to simplify interactions among our many varied entities and higher level systems.**
- And we don’t want to pay too much for that abstraction.

Examples of interactions:

- The main loop triggers an update telling everything that needs to be updated to update and everything that needs to be drawn to draw.
- During the update we see that Pacman has picked up a power-pellet. The pellet is consumed, a sound is played, the music changes, the score is updated, the ghosts switch to their fleeing states, a timer is started.

What is the goal?

- We want some kind of abstraction to simplify interactions among our many varied entities and higher level systems..
- **And we don’t want to pay too much for that abstraction.**

That cost can come in different forms:

- Maintenance
- Performance
- Flexibility

Let’s make a game

ShootyGuy: A fast-paced 2D arena fighting game with retro graphics and sound!

The player will face a variety of exciting enemies and obstacles:

MeleeGuy: Chases and attacks player in front of him when in range.

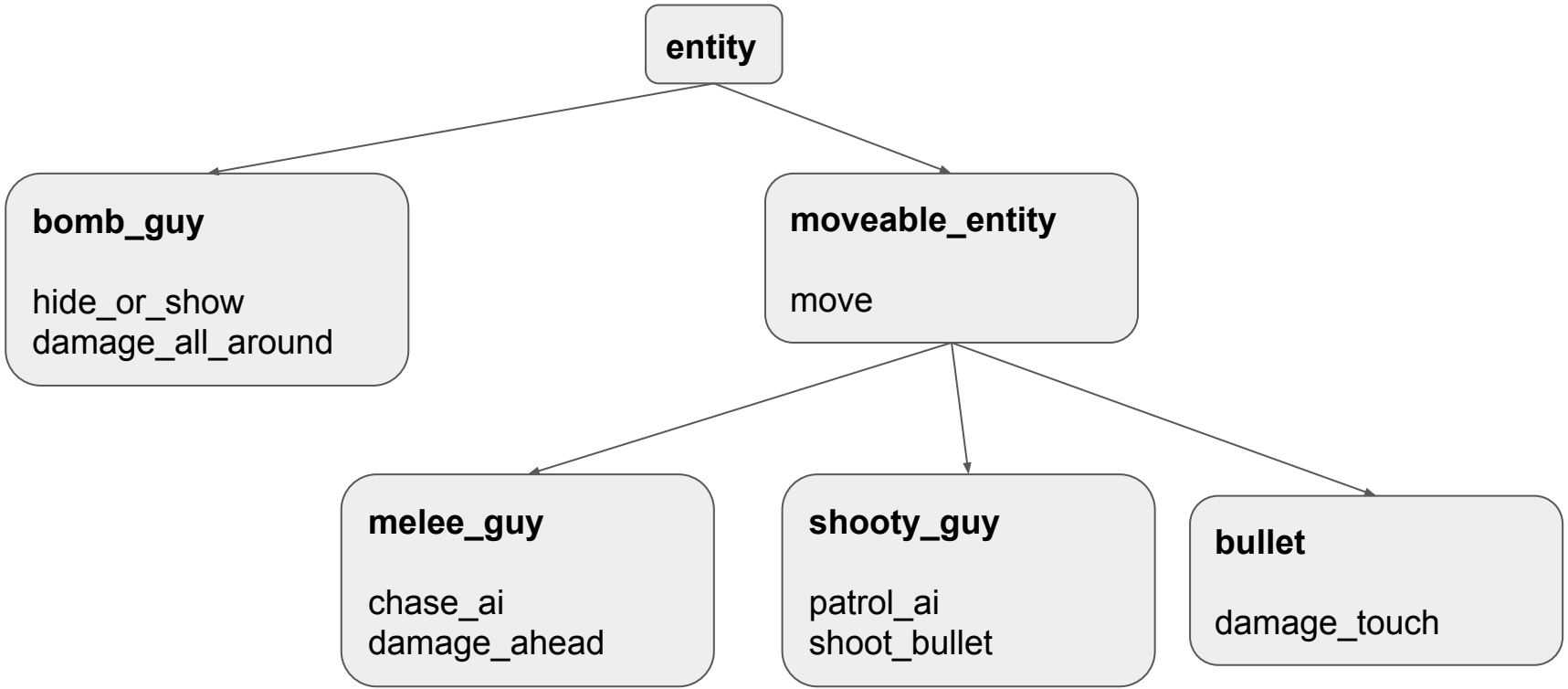
ShootyGuy: Patrols around and shoots at the player when seen.

BombGuy: Hides until player nearby and then explodes after a short delay.

The ShootyGuy entity model (Inheritance)

- In C++ the goto strategy for modeling relationships between closely related types is inheritance.
- And we can take advantage of polymorphism to give us some of the generic interaction we're after (i.e. abstract *update* and *draw* methods).

The ShootyGuy entity model (Inheritance)



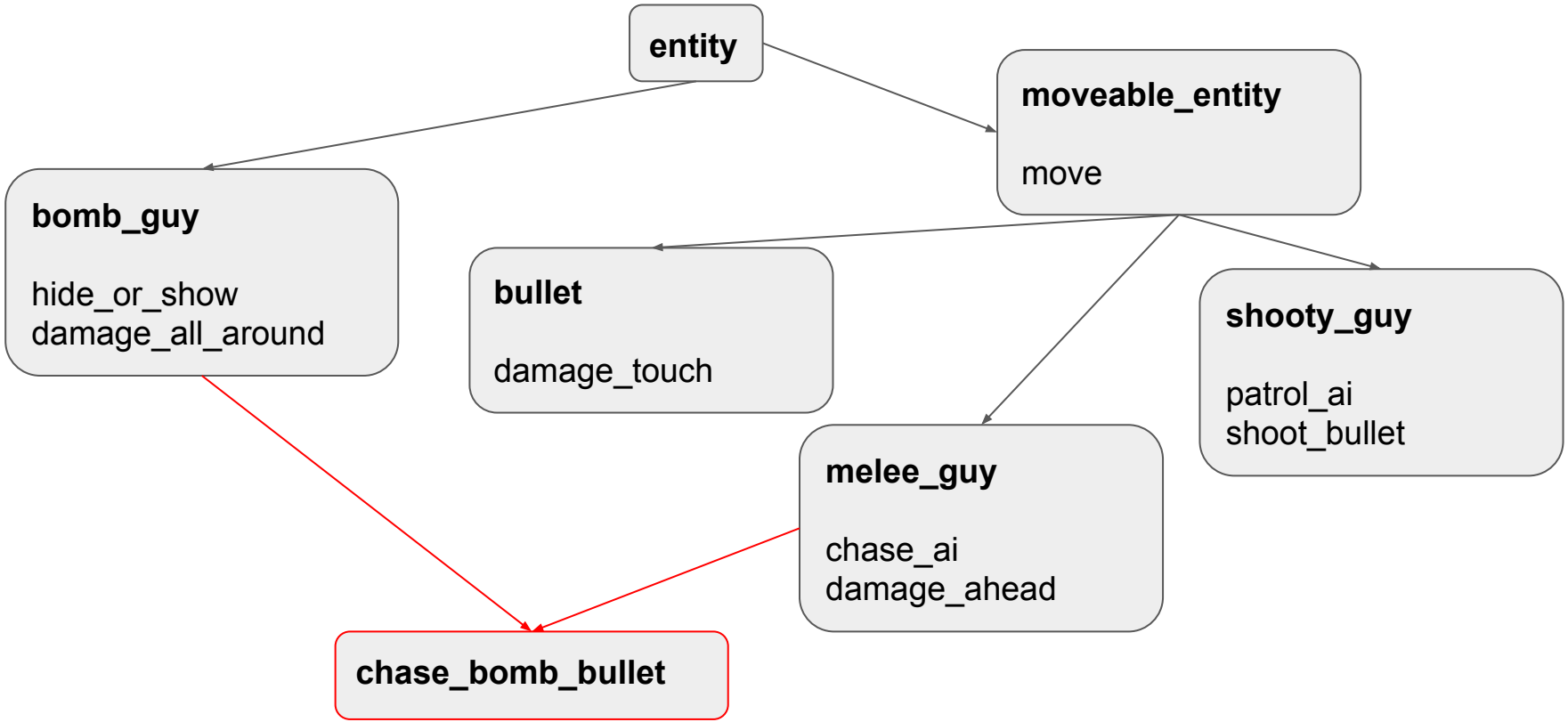
The ShootyGuy entity model (Inheritance)

```
struct ga_entity {
     virtual void update() = 0;
};

struct ga_bomb_guy : ga_entity
{
     void hide_or_show(bool show);
     void damage_all_around();

     void update() override
     {
          if (physics::is_player_nearby())
          {
               hide_or_show(true);
               damage_all_around();
          }
     }
};
```

Ship it!?

- What happens when your designer comes back and asks for a new bullet that chases the player like melee_guy and explodes like bomb_guy?

The ShootyGuy entity model (Inheritance)



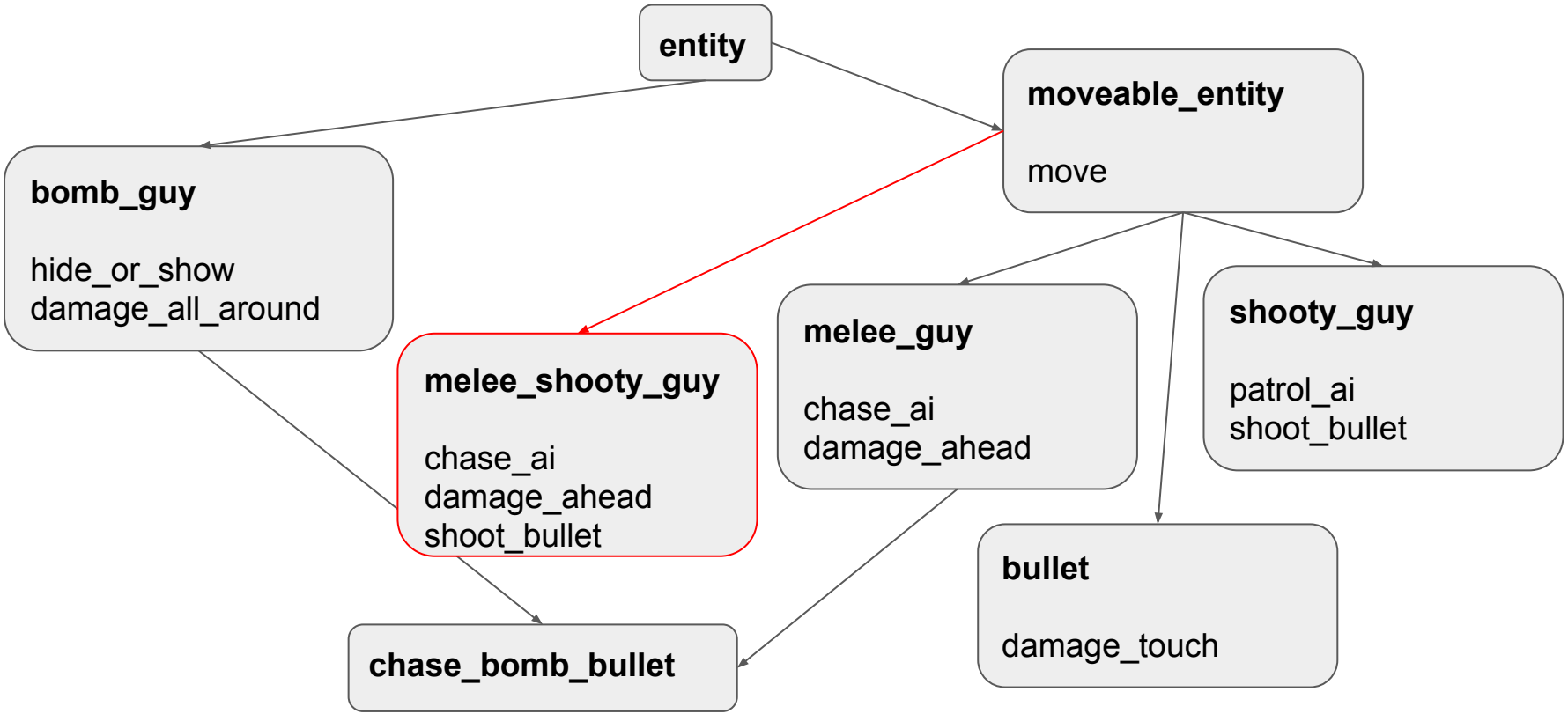
The diamond of death

```
struct ga_entity {
     virtual void update() = 0;
     virtual void draw() = 0;
};

struct ga_chase_bomb_bullet : ga_melee_guy, ga_bomb_guy {
     void update() override
     {
          // ?!?
          ga_moveable_entity::update();
          ga_bomb_guy::update();
     }

     void draw() override
     {
          // ?!?!?
          ga_moveable_entity::draw();
          ga_bomb_guy::draw();
     }
};
```

Ship it!?

- What happens when your designer comes back and asks for a new bullet that chases the player like melee_guy and explodes like bomb_guy?
- What about an enemy that can both shoot and punch?

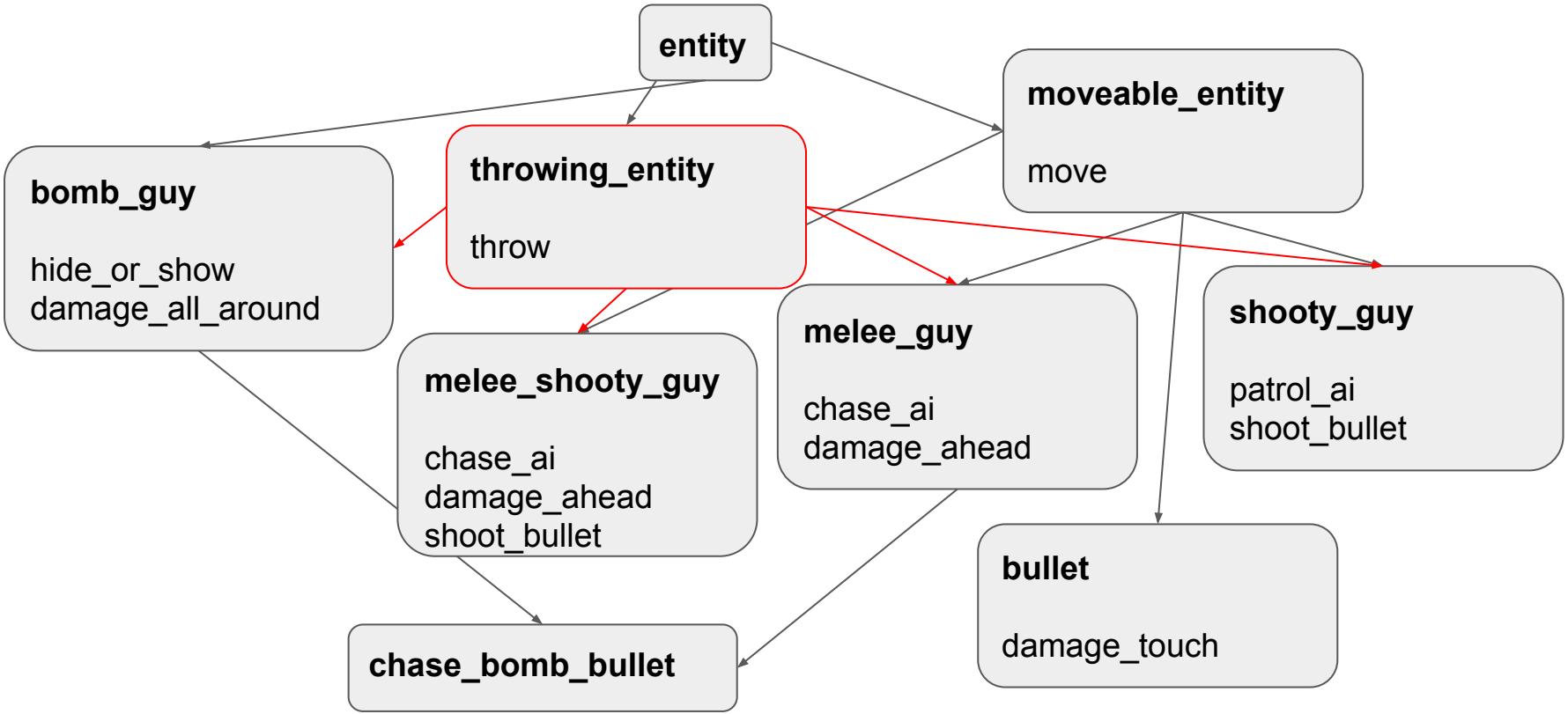
The ShootyGuy entity model (Inheritance)



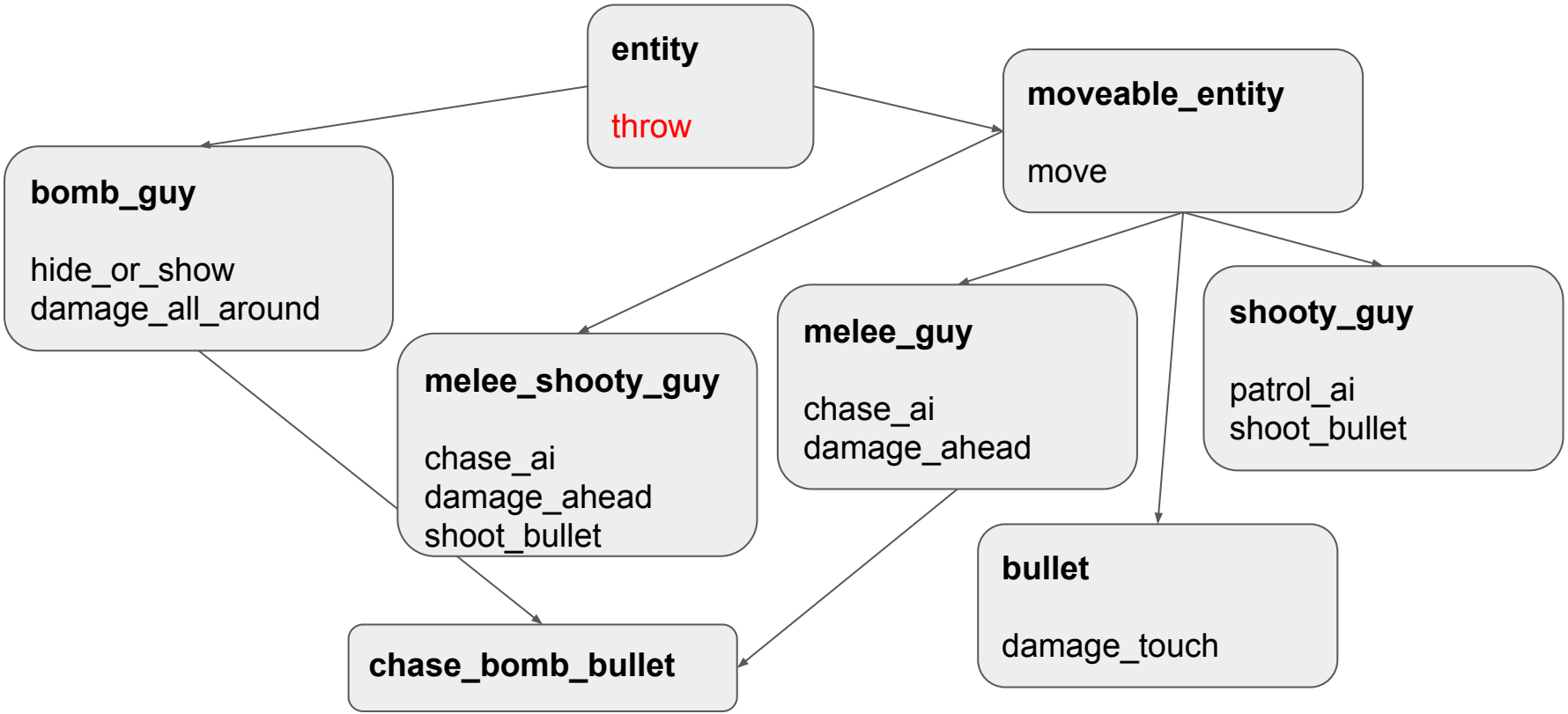
Ship it!?

- What happens when your designer comes back and asks for a new bullet that chases the player like melee_guy and explodes like bomb_guy?
- What about an enemy that can both shoot and punch?
- What about when we decide that we want all of our enemies to be able to pick up and throw rocks?

The ShootyGuy entity model (Inheritance)



The ShootyGuy entity model (Inheritance)



Other potential problems

- Functionality tends to drift upwards in the hierarchy. Or duplicated in leaf types.
  - Turns out it’s needed in places you didn’t anticipate!
- Need to understand the entire hierarchy to make changes to virtual methods.
  - That goes for both directions!
- Difficult to parallelize.
  - When you call *update* that means *everything* is going to update. Probably in a random order.
- Lots of virtual dispatch is poor for performance.
  - Constantly jumping to random locations in memory.

It can work though!

ai_ally_managerai_battle_line
ai_changehintgroup
ai_changetarget
ai_citizen_response_system
ai_goal_actbusy
ai_goal_actbusy queue
ai_goal_assault
ai_goal_follow
ai_goal_lead
ai_goal_lead weapon
ai_goal_police
ai_goal_standoff
ai_relationship
ai_script_conditions
ai_sound
ai_speechfilter
aiscripted_schedule
assault_assaultpoint
assault_rallypoint
info_apc_missile_hint
info_node
info_node_air
info_node_air hint
info_node_climb
info_node_hint
info_node_link
info_node_link_controller
info_npc spawn_destination
info_snipertarget
logic_choreographed_scene
path_corner
path_corner_crash
path_track
scripted_scene
scripted_sentence
scripted_sequence
scripted_target
tanktrain_aitarget
tanktrain_ai
env_ar2explosion
env_beam
env_beverage
env_blood
env_bubbles
Env_citadel_energy_core
env_credits
env_cubemap
env_dustpuff
env_effectscript
env_embers
env_entity_dissolver
env_entity_igniter
env_entity_maker
env_explosion
env_extinguisherjet
env_fade
env_fire
env_firesensor
env_firesource
env_flare
env_fog_controller
env_funnel
env_global
env_gunfire
env_headcrabcanister
env_hudhint
env_laser
env_lightglow
env_message
env_microphone
env_muzzleflash
env_particlelight
env_particlescript
env_physexplosion
env_physimpact
env_player_surface_trigger
env_rotorshooter
env_rotorwash
env_screenoverlay
env_shake
env_shooter
env_smokestack
env_smoketrail
env_soundscape
env_soundscape_proxy
env_soundscape_triggerable
env_spark
env_speaker
env_splash
env_sprite
env_spritetrail
env_starfield

It can work though! Sorta.

env_steam
env_sun
env_terrainmorph
env_texturetoggle
env_tonemap_controller
env_wind
env_zoom
filter_activator_class
filter_activator_name
filter_activator_team
filter_damage_type
filter_multi
func_areaportal
func_areaportalwindow
func_breakable
func_breakable_surf
func_brush
func_button
func_capturezone
func_changeclass
func_clip_vphysics
func_combine_ball_spawner
func_conveyor
func_detail
func_door
func_door_rotating
func_dustcloud
func_dustmotes
func_extinguishercharger
func_guntarget
func_healthcharger
func_illusionary
func_ladder
func_ladderendpoint
func_lod
func_lookdoor
func_monitor
func_movelinear
func_nobuild
func_nogrenades
func_occluder
func_physbox
func_physbox_multiplayer
func_platrot
func_precipitation
func_proprespawnzone
func_recharge
func_reflective_glass
func_regenerate
func_respawnroom
func_respawnroomvisualizer
func_rot_button
func_rotating
func_smokevolume
func_tank
func_tankairboatgun
func_tankapcrocket
func_tanklaser
func_tankmortar
func_tankphyscannister
func_tankpulselaser
func_tankrocket
func_tanktrain
func_trackautochange
func_trackchange
func_tracktrain
func_traincontrols
func_useableladder
func_vehicleclip
func_viscluster
func_wall
func_wall_toggle
func_water_analog
worldspawn
game_end
game_player_equip
game_player_team
game_ragdoll_manager
game_score
game_text
game_ui
game_weapon_manager
game_zone_player
info_camera_link
info_constraint_anchor
info_hint
info_intermission
info_ladder_dismount
info_landmark
info_lighting
info_mass center
Info_no_dynamic_shadow

It can work though! Sorta =(

nfo_node
info_node_air
info_node_air hint
info_node_climb
info_node_hint
info_node_link
info_node_link_controller
info_npc_spawn_destination
info_null
info_overlay
info_particle_system
info_player_combine
info_player_deathmatch
info_player_logo
info_player_rebel
info_player_start
info_projecteddecal
info_snipertarget
info_target
info_target_gunshipcrash
info_teleporter_countdown
info_teleport_destination
infodecal

item_ammo_357
item_ammo_357_large
item_ammo_ar2
item_ammo_ar2_altfire
item_ammo_ar2_large
item_ammo_crate
item_ammo_crossbow
item_ammo_pistol
item_ammo_pistol_large
item_ammo_smg1
item_ammo_smg1_grenade
item_ammo_smg1_large
item_battery
item_box_buckshot
item_dynamic resupply
item_healthcharger
item_healthkit
item_healthvial
item_item_crate
item_rpg_round
item_suit
item_suitcharger
light

light_spot
light_dynamic
env_projectedtexture
point_spotlight
light_environment
light_directional
logic_auto
logic_autosave
logic_branch
logic_case
logic_collision_pair
logic_compare
logic_lineto
logic_measure_movement
logic_multicompare
logic_navigation
logic_relay
logic_timer
ambient_generic
cycler
gibshooter
keyframe_rope
keyframe_track

material_modify_control
math_colorblend
math_counter
math_remap
momentary_rot_button
move_keyframed
move_rope
move_track
script_intro
script_tauremoval
shadow_control
sky_camera
test_sidelist
test_traceline
vgui_screen
water_lod_control
combine_mine
npc_alyx
npc_antlion
npc_antlion_template_maker
npc_antlionguard
npc_barnacle
npc_barney

It can work though! Sorta =((

npc_breen
npc_citizen
npc_combine_camera
npc_combine_s
npc_combinedropship
npc_combinegunship
npc_crabsynth
npc_cranedriver
npc_crow
npc_cscanner
npc_dog
npc_eli
npc_fastzombie
npc_fisherman
npc_gman
npc_headcrab
npc_headcrab_black
npc_headcrab_fast
npc_helicopter
npc_ichthyosaur
npc_kleiner
npc_manhack
npc_metropolice
npc_monk
npc_mortarsynth
npc_mossman
npc_pigeon
npc_poisonzombie
npc_rollermine
npc_seagull
npc_sniper
npc_stalker
npc_strider
npc_turret_ceiling
npc_turret_floor
npc_turret_ground
npc_vortigaunt
npc_zombie
npc_zombie_torso
cycler_actor
generic_actor
info_npc_spawn_destination
monster_generic
npc_apcdriver
npc_bullseye
npc_enemyfinder
npc_furniture
npc_heli_avoidbox
npc_heli_avoidsphere
npc_heli_nobomb
npc_launcher
npc_maker
npc_missiledefense
npc_particlestorm
npc_spotlight
npc_template_maker
npc_vehicledriver
phys_ballsocket
phys_constraint
phys_constraintsystem
phys_convert
phys_hinge
phys_keepupright
phys_lengthconstraint
phys_magnet
phys_motor
phys_pulleyconstraint
phys_ragdollconstraint
phys_ragdollmagnet
phys_slideconstraint
phys_spring
phys_thruster
phys_torque
physics_cannister
player_loadsaved
player_speedmod
player_weaponstrip
point_anglesensor
point_angularvelocitysensor
point_antlion_repellant
point_apc_controller
point_bugbait
point_camera
point_clientcommand
point_commentary_node
point_devshot_camera
point_enable_motion_fixup
point_hurt
point_message
point_playermoveconstraint
point_servercommand
point_spotlight

It can work though! Sorta =(((

point_teleport
point_template
point_tesla
point_viewcontrol
vehicle_viewcontroller
prop_combine_ball
prop_detail
prop_door_rotating
prop_dynamic
prop_dynamic_ornament
prop_dynamic_override
prop_physics
prop_physics_multiplayer
prop_physics_override
prop_ragdoll
prop_static
prop_thumper
prop_vehicle
prop_vehicle_airboat
prop_vehicle_apc
prop_vehicle_cannon
prop_vehicle_crane
prop_vehicle_driveable
prop_vehicle_jeep
prop_vehicle_prisoner_pod
trigger_autosave
trigger_changelevel
trigger_finale
trigger_gravity
trigger_hurt
trigger_impact
trigger_look
trigger_multiple
trigger_once
trigger_physics_trap
trigger_playermovement
trigger_proximity
trigger_push
trigger_remove
trigger_rpgfire
trigger_soundscape
trigger_teleport
trigger_transition
trigger_vphysics_motion
trigger_waterydeath
trigger_weapon_dissolve
trigger_weapon_strip
trigger_wind
weapon_357
weapon_alyxgun
weapon_annabelle
weapon_ar2
weapon_brickbat
weapon_bugbait
weapon_citizenpackage
weapon_citizensuitcase
weapon_crossbow
weapon_crowbar
weapon_extinguisher
weapon_frag
weapon_physcannon
weapon_physgun
weapon_pistol
weapon_rpg
weapon_shotgun
weapon_smg1
weapon_stunstick

Review

- Inheritance helped in some ways but we've sacrificed **a-lot** of flexibility.
  - A large portion of our game design is now hard-coded in C++!
- Everything you do will eventually be wrong. Need to be able to quickly adapt!

Wisdom

There are probably hundreds of ways you could decompose your systems and come up with a set of classes and, eventually, all of them are wrong. This isn’t to say that they won‟t work, but games are constantly changing, constantly invalidating your carefully planned designs.

So you hand off your new Game Object System and go work on other things. Then one day your designer says that they want a new type of “alien” asteroid that acts just like a heat seeking missile, except it‟s still an asteroid. Or they want to get rid of this whole spaceship concept and go underwater instead...

A Data Driven Game Object System
GDC 2012
Scott Bilas

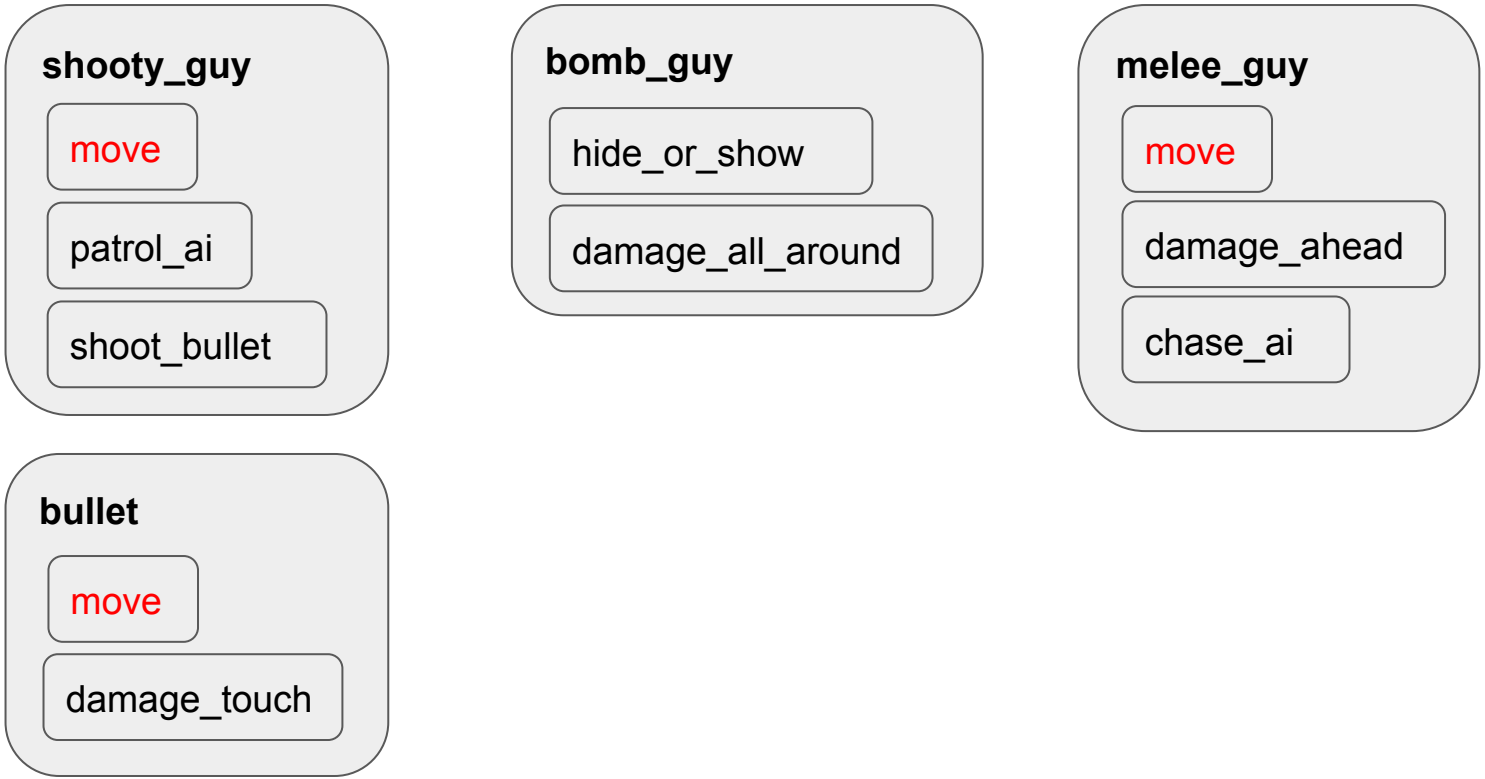
If not inheritance, than what?

- Inheritance represents an “is-a” relationship.
  - A square is a shape.
  - A duck is an animal.
  - The melee_guy is a moveable_entity

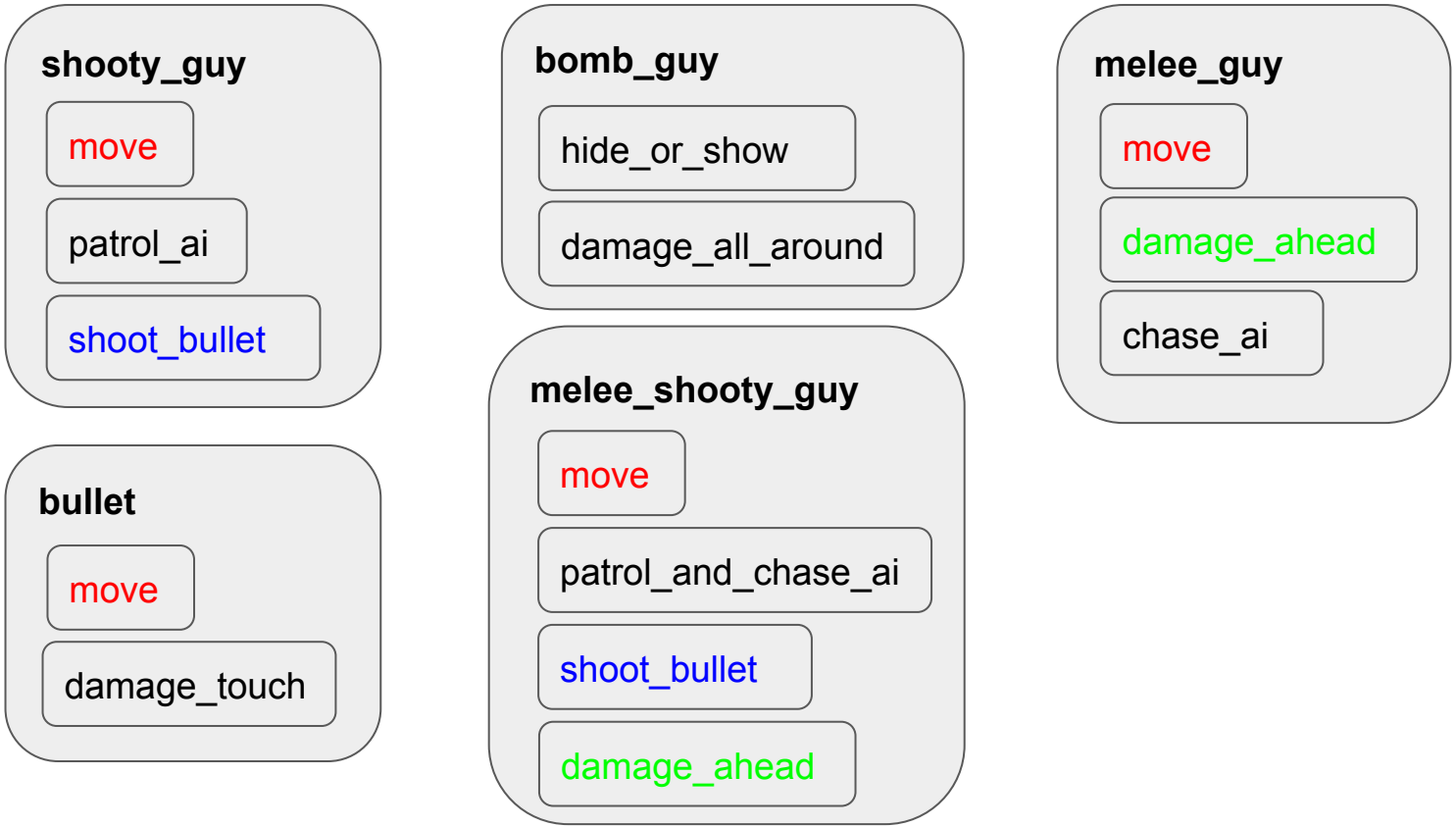
If not inheritance, than what?

- Inheritance represents an “is-a” relationship.
  - A square is a shape.
  - A duck is an animal.
  - The melee_guy is a moveable_entity
- Composition represents a “has-a” relationship.
  - A car has an engine.
  - The employee has a name.
  - The melee_guy has movement and melee attacks.

The ShootyGuy entity model (Composition)



The ShootyGuy entity model (Composition)



The ShootyGuy entity model (Composition)

```
struct ga_move final {...};
struct ga_patrol_ai final {...};
struct ga_shoot_bullet final {...};
struct ga_chase_ai final {...};
struct ga_damage_ahead final {...};

struct ga_shooty_guy final
{
     ga_move* _move;
     ga_patrol_ai* _patrol_ai;
     ga_shoot_bullet* _shoot_bullet;
};

struct ga_melee_guy final
{
     ga_move* _move;
     ga_chase_ai* _chase_ai;
     ga_damage_ahead* _damage_ahead;
};
```

The ShootyGuy entity model (Components)

```
struct ga_component
{
	virtual void update() = 0;
};
struct ga_move_component : ga_component {...};
struct ga_patrol_ai_component : ga_component {...};
struct ga_shoot_bullet_component : ga_component {...};

struct ga_entity final
{
	std::vector<ga_component*> _components;

	void update()
	{
		for (auto& component : _components)
		{
			component->update();
		}
	}
};
```

The ShootyGuy entity model (Components)

```
{
	ga_entity shooty_melee_guy;

	shooty_melee_guy.add_component(new ga_move_component);
	shooty_melee_guy.add_component(new ga_patrol_and_chase_ai_component);
	shooty_melee_guy.add_component(new ga_shoot_bullet_component);
	shooty_melee_guy.add_component(new ga_damage_ahead_component);

	ga_sim sim;

	sim.add_entity(&shooty_melee_guy);

	while (true) {
		sim.update();
	}
}
```

Recap

- No longer need to change entity hierarchy to accommodate gameplay changes!

Recap

- No longer need to change entity hierarchy to accommodate gameplay changes!
- Entities are built at runtime. This could allow us to load specifications from data so they can be changed/created by designers without recompiling code.

Recap

- No longer need to change entity hierarchy to accommodate gameplay changes!
- Entities are built at runtime. This could allow us to load specifications from data so they can be changed/created by designers without recompiling code.
- Easier to maintain since (ideally) components are independent of each other and can be understood in isolation.

Can we do any better?

- What we have is very flexible and similar approaches have been used to ship many modern games.
- But… the performance is still not great. Perhaps even worse than the inheritance based approach.
    - Even more virtual method calls (for each component instead of each entity).
    - Need to search list for correct component if specific functionality is needed.

Entities as entries in database

- In a component based system the entity is little more than an interface for querying data in its components.
- Which sounds a lot like a database!
- A database that we've optimized for a single very specific type of query.
    - Find all the components associated with this entity.
- But is that the most useful or common query we might want to make?
- What are some other queries?
    - Find all the move components.
    - Find all the move components that are attached to entities that also have physics components.

Entity Database

```
using ga_entity = uint32_t;

struct ga_move_component { void move(); };

struct ga_entity_database final {
     std::array<ga_move_component, k_max_components> _move_components;

     ga_entity create_entity() {
          static next_entity_id = 0;
          return next_entity_id++;
     }

     ga_move_component& get_move_component(ga_entity entity) const {
          return _move_components[entity];
     }

     void update() {
          for (auto& component : _move_components) { component.move(); }
     }
};
```

Other consideration

- Components become POD. No methods.
- All of the methods are moved into *systems* that operate on components.
- A system need not operate on one type of component at a time.
- Dependencies become more obvious. The physics system needs the shape component and the transform component. No other interactions are allowed.
- Data can be interleaved together in memory to maximize performance.

Questions?