



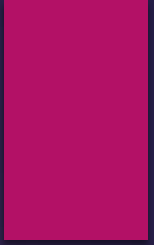
# An Introduction to CQRS

# Who Am I?

- ▶ Web Engineer for Celina Insurance Group.
- ▶ Writing software for over a decade. The majority of which has been web applications.
- ▶ Twitter: @sam\_ferree
- ▶ [dev.to/sam\\_ferree](https://dev.to/sam_ferree)
- ▶ [github.com/CzechsMix/talks/introduction-to-cqrs](https://github.com/CzechsMix/talks/introduction-to-cqrs)

# What is CQRS

- ▶ Command Query Responsibility Segregation
- ▶ Coined by Greg Young in 2010
- ▶ Comes From Bertrand Myer's CQS



Asking a question  
shouldn't  
change the  
answer.

```
public class Attendee
{
    → public Guid Id { get; set; }
    → public string Name { get; set; }
    → public string Email { get; set; }

    → // Lazy Loaded via ORM
    → public List<Ticket> Tickets { get; set; }

    → public List<Event> Events =>
    → | Tickets.Select(ticket => ticket.Event).ToList();

    → public HasTicketFor(Guid eventId) =>
    → | Tickets.Any(ticket => ticket.EventId == eventId);
}
```

# Why Models?

```
public Result CreateTicket(Ticket ticket)
{
    → var attendee = _attendees.Find(ticket.AttendeeId);

    → if (attendee is null)
    → → return Result.Error("Attendee does not exist.");

    → if (attendee.HasTicketFor(ticket.EventId))
    → → return Result.Error("Attendee has already purchased ticket for this event.");

    → _tickets.Create(ticket);

    → return Result.Success;
}
```

@sam\_ferree

# Domain Logic

```
public Result UpdateTicket(Ticket ticket)
{
    → var existing = _tickets.Find(ticket.Id);

    → if (existing is null)
    → → return Result.Error("Ticket does not exist.");

    → if (existing.AttendeeId != ticket.AttendeeId)
    → {
    → → if (existing.Redeemed)
    → → → return Result.Error("Cannot transfer Redeemed Ticket.");

    → → var newAttendee = _attendees.Find(ticket.AttendeeId);
    → → if (newAttendee is null)
    → → {
    → → → return Result.Error("New Attendee does not exist.");
    → → }
    → }

    → // apply any permitted changes
    → existing.Redeemed = ticket.Redeemed;
    → existing.AttendeeId = ticket.AttendeeId;

    → _tickets.Update(existing);

    → return Result.Success;
}
□
```

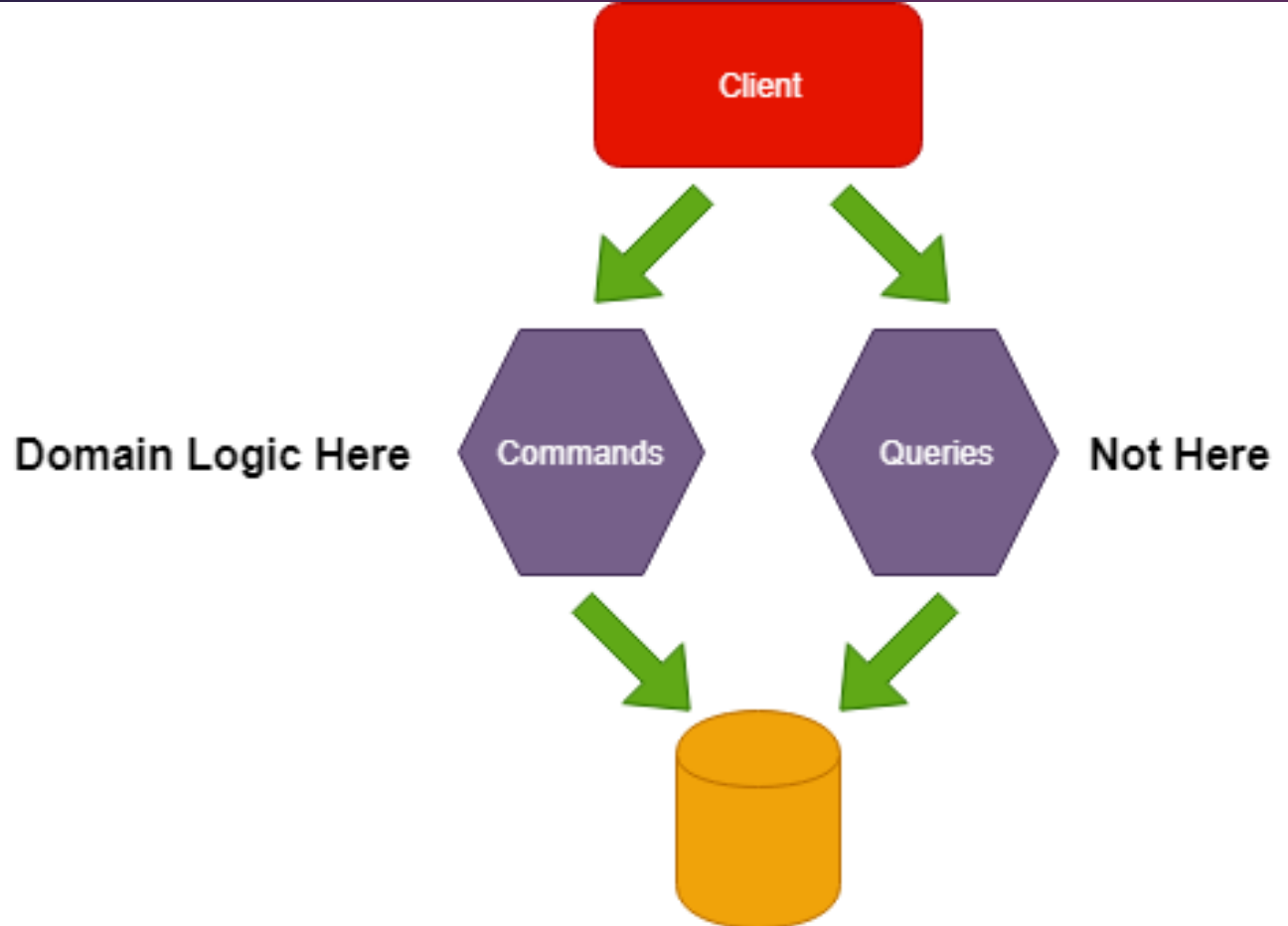
Updates  
can do  
too  
much

```
public List<Attendee> AttendeesByEventsAttended(int eventsAttended)
{
    → return _attendees.Where(attendee =>
    → | attendee.Tickets.Where(ticket =>
    → | | ticket.Redeemed).Count() >= eventsAttended).ToList();
}
```

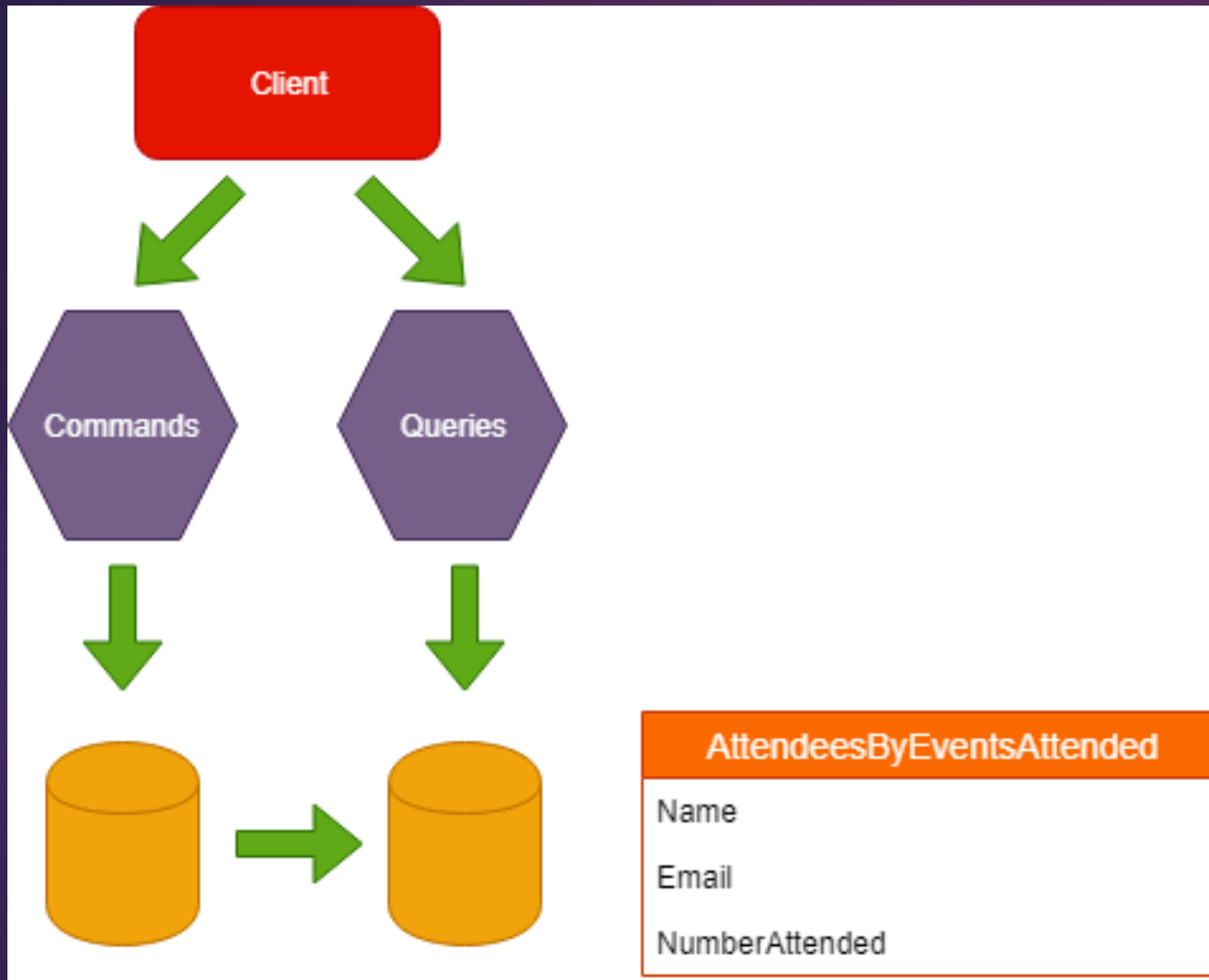
@sam\_ferree

# Reads are deceptive





Queries  
are logic  
free



# Querying separate databases

```
public Result RedeemTicket(RedeemTicketCommand command)
{
→ var ticket = _tickets.Find(command.ticketId);

→ if (ticket is null)
→ | return Result.Error("Ticket does not exist.");

→ if (ticket.Redeemed)
→ | return Result.Error("Ticket already redeemed");
→
→ ticket.Redeemed = true;
→ _tickets.Update(ticket);

→ return Result.Success;
}
```

# Redeem Command

```
public Result TransferTicket(TransferTicketCommand command)
{
    → var ticket = _tickets.Find(command.ticketId);

    → if (ticket is null)
    → | → return Result.Error("Ticket does not exist.");

    → if (ticket.Redeemed)
    → | → return Result.Error("Cannot Transfer already redeemed ticket.");

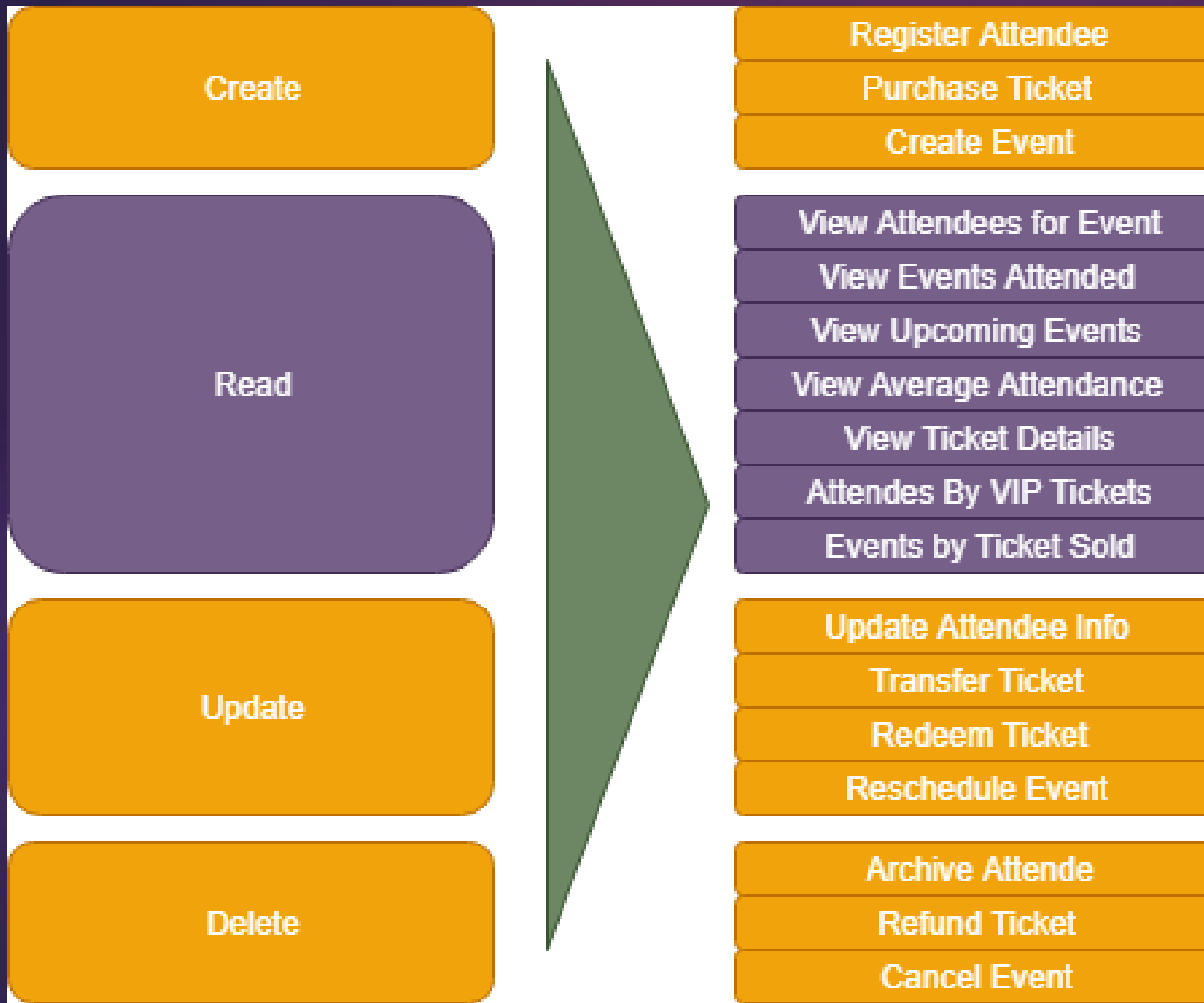
    → var newAttendee = _attendees.Find(command.newAttendeeId);

    → if (newAttendee is null)
    → | → return Result.Error("Transfer recipient does not exist.");
    →
    → if (newAttendee.HasTicketFor(ticket.EventId))
    → | → return Result.Error("Transfer recipient already has ticket for event.");

    → ticket.AttendeeId = newAttendee.Id;
    → _tickets.Update(ticket);

    → return Result.Succes;
}
```

# Transfer Command



# CRUD- Based vs Task Based

```
→ [HttpPost("/Tickets/{ticketId}/Redeem")]
→ public IActionResult RedeemTicket(Guid ticketId)
→ {
→     var result = _mediator.Send(new RedeemTicketCommand(ticketId));
→
→     return Ok(result);
→ }

public class RedeemTicketHandler : CommandHandler<RedeemTicketCommand>
{
→ public Result Handle(RedeemTicketCommand command)
→ {
→     var ticket = _tickets.Find(command.ticketId);

→     if (ticket is null)
→     | return Result.Error("Ticket does not exist.");

→     if (ticket.Redeemed)
→     | return Result.Error("Ticket already redeemed");

→     ticket.Redeemed = true;
→     _tickets.Update(ticket);

→     return Result.Success;
→ }
}
```

# The Mediator Pattern

# Trade Offs

- ▶ Complexity
- ▶ Eventual Consistency (aka Stale Data)

# When to use CQRS

- ▶ Collaborative Domain
- ▶ Many and/or Complex Queries
- ▶ Large Complex Monolith
- ▶ Event Sourcing



# Where to next?

- ▶ Pluralsight Courses:
  - ▶ <https://app.pluralsight.com/library/courses/modern-software-architecture-domain-models-cqrs-event-sourcing/table-of-contents>
  - ▶ <https://app.pluralsight.com/library/courses/cqrs-in-practice/table-of-contents>
- ▶ Jimmy Bogard's Contoso University:
  - ▶ <https://jimmybogard.com/contoso-university-examples-with-cqrs-mediator-automapper-and-more/>
- ▶ An Introduction to CQRS and Event Sourcing Patterns - Mathew McLoughlin:
  - ▶ <https://www.youtube.com/watch?v=9a1PqwFrMP0>
- ▶ [github.com/CzechsMix/talks/introduction-to-cqrs](https://github.com/CzechsMix/talks/introduction-to-cqrs)

The background is a dark purple gradient. On the left, there are several concentric, faint purple circles. A thin, light purple vertical line is positioned to the left of the word 'Questions?'. In the top right corner, there is a solid pink rectangle.

# Questions?

@sam\_ferree