1 Typed Subgroups
   - Remote Procedure Calls
   - State Machine Replication
   - Derecho as an SMR library
   - Implementation
Raw vs Typed Subgroups

- Raw subgroups have no state. They just provide functionality to send and deliver raw messages.
- Typed subgroups are instances of C++ classes written by users. They have an associated state (class data members).
- They allow sending what we call **cooked messages**. Cooked messages, when delivered in a subgroup, result in invocations of the subgroup’s state operations. These state operations are just the subgroup class’s member functions.
- A cooked message carries information about the function to call at delivery and the arguments to pass to it.
Remote Procedure Calls (RPCs)

- RPCs are a way to run a function at a remote server (and get the results back).
- Cooked messages are a way to implement RPCs.
- RPCs involve the following steps:
  - Serializing the arguments into a buffer (also called marshaling)
  - Sending the buffer over the network to the remote node
  - Deserializing the received buffer to obtain the arguments (demarshaling) and the function tag at the remote node
  - Calling the requested function with the arguments and getting back the results (the return value)
  - Sending the results back to the calling node
A state machine is something that changes its state as a result of external commands.

As an example, a ticket booking server responds to external customer commands to book or cancel tickets. Its state is the information about which tickets are booked and which are still available for booking.

A single state machine is prone to failures. If my server crashes (many possible reasons: segfault, power failure, natural disaster etc.), then the ticket booking service is down.
State Machine Replication (SMR)

- To keep my service running, I can create replicas of my machine that all start from the same state and process all the commands in the same order.
- Thus, every server will book and cancel the same tickets. This guards against failures. If one of them fails, others will keep running.
- This technique is called State Machine Replication.
Can you see now how Derecho provides SMR?

- Every Derecho subgroup is a replicated state machine with its members representing the individual machines.
- Its state is simply its C++ class’s object state (which Derecho guarantees to be the same across all the members).
- Its state operations (or commands) are its member functions.
- Derecho’s RPC layer enable calling the state operations at the members. Derecho’s delivery layer ensures that the state operations are called in the same order at all the members.
Example

- In our ticket booking server example, customer requests reach one of the server nodes first. Then the servers send a message in the subgroup so that every node carries the operation.

- Thus, if node 0 gets a book (t1) call, node 1 gets a book (t2) call and node 2 gets a cancel (t1) call, all nodes run book(t1), book(t2), cancel(t1) in the same order.

- This guarantees strong consistency: Every subgroup member maintains the same state over time. If a book operation returns error at one node, it returns error at all the nodes. If a cancel operation succeeds at one node, it succeeds at all the nodes.
Consistency with subgroup membership changes

- Node failures can lead to changes in subgroup membership. We can also add more nodes (for example, to compensate for failed nodes).
- If a new node joins, it lacks the current state of the subgroup! It needs to get that state before it can execute the same operations with everyone else.
- This is called state transfer.
- Similarly, when a subgroup starts for the first time (no previous state), the initialization of the associated C++ class object should lead to the same starting state for all the members.
How to write typed subgroups with Derecho?

- Write the subgroup classes with data members for the state and member functions for the desired operations.
- Specify the data members that constitute the state. In your class, add a line (e.g.):

```cpp
default_serialization_support(
    TicketBookingSystem, num_tickets, booked);
```

Have your class inherit publicly from `mutils::ByteRepresentable` (for serialization). Like this:

```cpp
class TicketBookingSystem : public mutils::ByteRepresentable {
```
How to write typed subgroups with Derecho?

- Register the RPC functions with Derecho. Another line in your class, like this:

```cpp
REGISTER_RPC_FUNCTIONS(TicketBookingSystem, book, cancel);
```

- Tell Derecho how to construct an object of your subgroup. Write a function that constructs such an object (called a factory).

```cpp
auto ticket_subgroup_factory = [] (PersistentRegistry*) { return std::make_unique<TicketBookingSystem>(15);};
```
How to write typed subgroups with Derecho?

- Pass this factory as one of the group constructor parameters
- Template the group on the type of the subgroup
- Write a membership function for your typed subgroup that tells Derecho how to assign members to your subgroup
Calling Subgroup operations

- As always, first obtain a handle for the subgroup from the group
- Call ordered_query or ordered_send on it. Specify the name of the function in the template and the function arguments in the arguments. Like this:

```cpp
1  ticketBookingHandle.ordered_query<RPC_NAME(book)>(my_rank);
```

- Next meeting: How to get the function return value!