DARTMOUTH

Automatic Schema Inference from Unknown Protobuf Messages

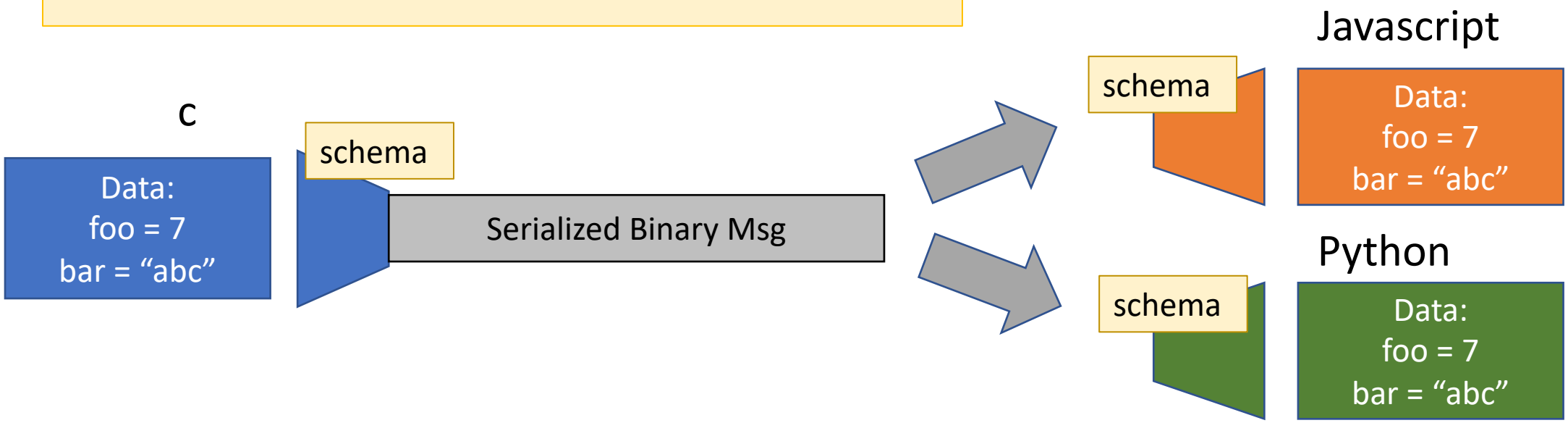
Jared Chandler

This work results from the SPLICE research program, supported by a collaborative award from the National Science Foundation (NSF) SaTC Frontiers program under award number 1955805.

Protocol buffers (protobuf)

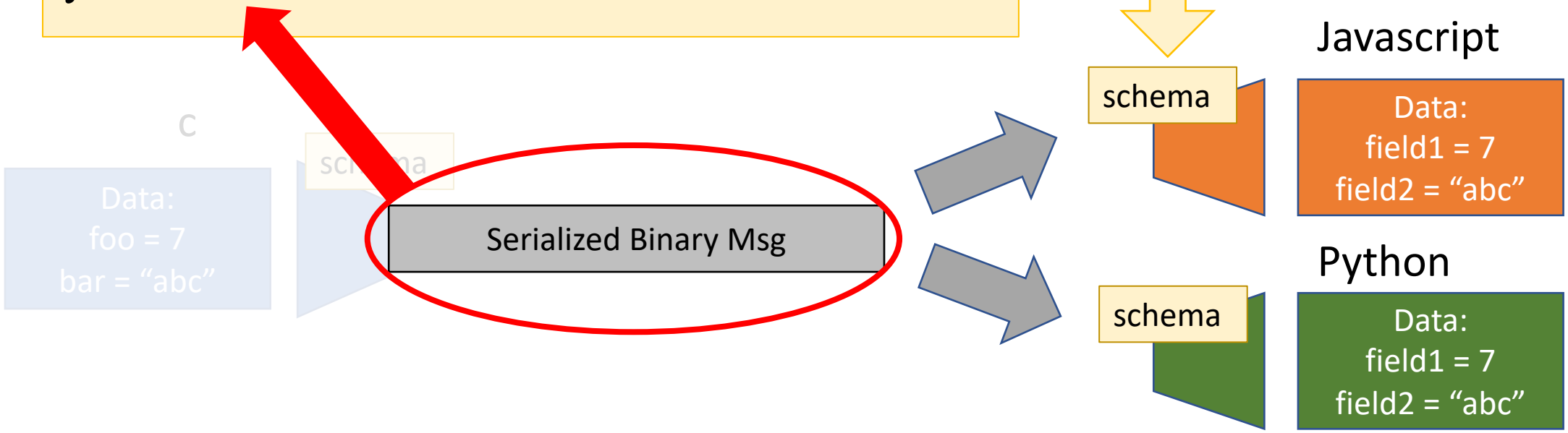
```
message Example {
   required int32  foo = 1;
   repeated string bar = 2;
}
```

- Developed by Google
- Serialized Data Middleware



Protocol buffers (protobuf)

```
message Example {
   required int32  field1 = 1;
   repeated string field2 = 2;
}
```

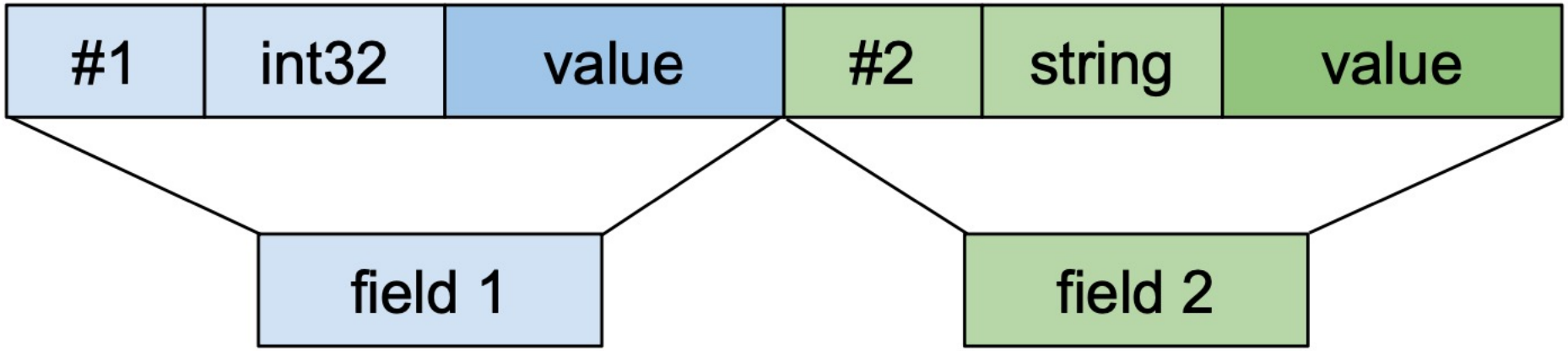


```
[08] 1 varint: 953643231 (0x38d770df)
[10] 2 varint: 1600177418 (0x5f60c50a)
[1a] 3 string: (2385):
    [ba 05] 87 string: (5): wlan0 (77 6c 61 6e 30)
    [08] 1 varint: 0 (0x0)
    [10] 2 varint: 1 (0x1)
    [18] 3 varint: 0 (0x0)
    [22] 4 string: (32): acc137f0717aa67b2ab7347506aa07ed
    [2a] 5 string: (53): MPSS.AT.3.1-00777-SDM660_GEN_PACK-1.260989.1.267581.1
    [32] 6 string: (1): 9 (39)
    [3a] 7 string: (15): 868042031
    [42] 8 string: (16): 5b651fce2
    [4a] 9 string: (7): ac7887a (6
    [52] 10 string: (5): MI 6X (4d
        [4d] 9 fix32/float: 147994
    [58] 11 varint: 8 (0x8)
    [62] 12 string: (33): Qualcomm
    [6a] 13 string: (42): fp asimd
    [72] 14 string: (0):
    [7a] 15 string: (14): <unknown
    [82 01] 16 string: (17): 02:00
```

```
1: 1
2: "497C49D8-86BD-4FED-A1EE-C22F45466E9F"
3: 0x41c16b3da7808a2f
6 {
  1: "Wegmans"
  2: "Fairfax"
  4 {
    5: 0x40435d6271340000
    6: 0xc0535cbae0f40000
    7: 0x40437fa5cb740000
    8: 0xc05346cd42000000
  }
}
11: 0
```

Self describing? So this is easy right?

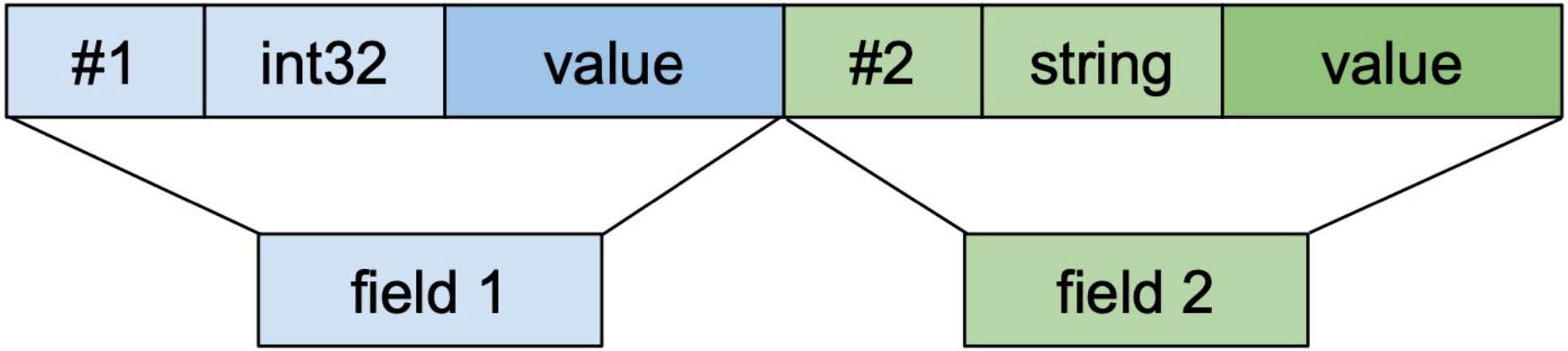
```
message Example {
    required int32  field1 = 1;
    repeated string field2 = 2;
}
```



Self describing? So this is easy right?

```
message Example {
    required int32  field1 = 1;
    repeated string field2 = 2;
}
```

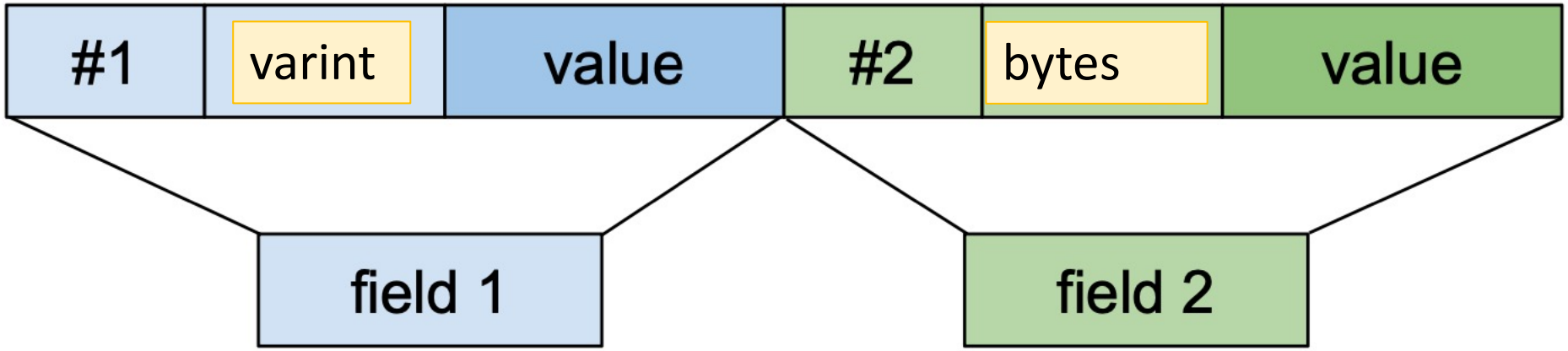
No Field Names



Self describing? So this is easy right?

```
message Example {
    required int32  field1 = 1;
    repeated string field2 = 2;
}
```

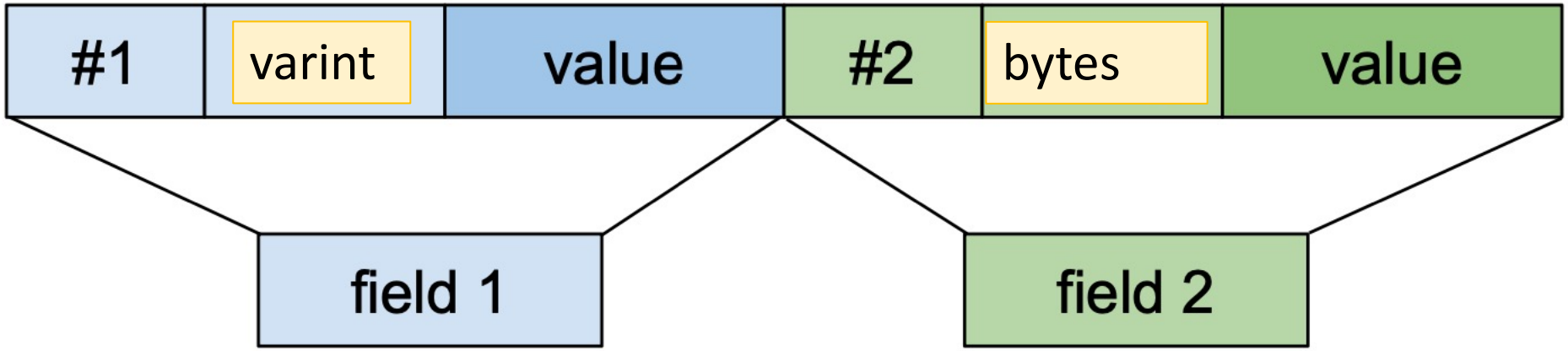
Wire Datatypes Are Approximations of Schema Types



Self describing? So this is easy right?

```
message Example {
    required int32  field1 = 1;
    repeated string field2 = 2;
}
```

No Arity Information



Self describing… with some caveats

🚫 No Arity Information

🚫 No Field Names

🤷‍♀️ Approximate Types of Fields

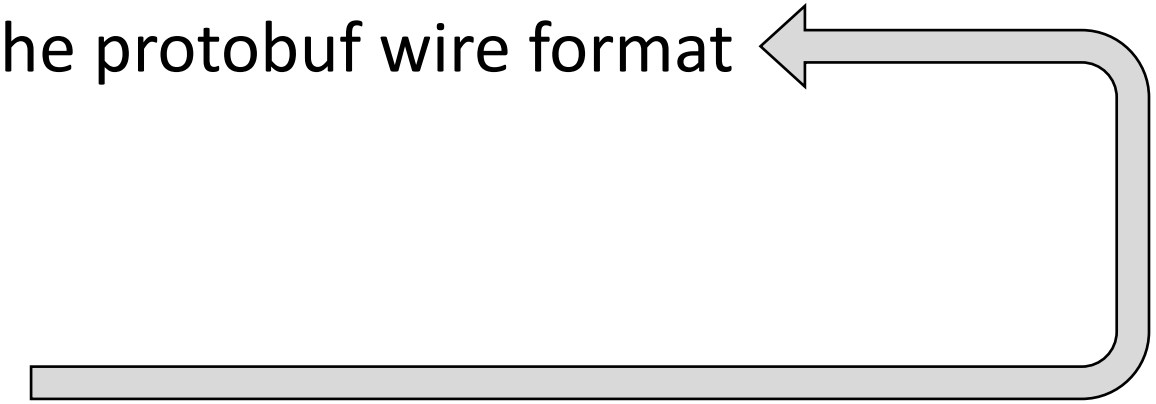
bytes → bytes | string
varint → int32 | uint32 | etc.

✅ Field Numbers

🤷‍♀️ Byte fields could be strings, or bytes or subformats

Schema Inference Methodology

1. Interpret messages according the protobuf wire format
2. Count Field Numbers
3. Infer Arities
4. Recurse onto Byte Field *Values*



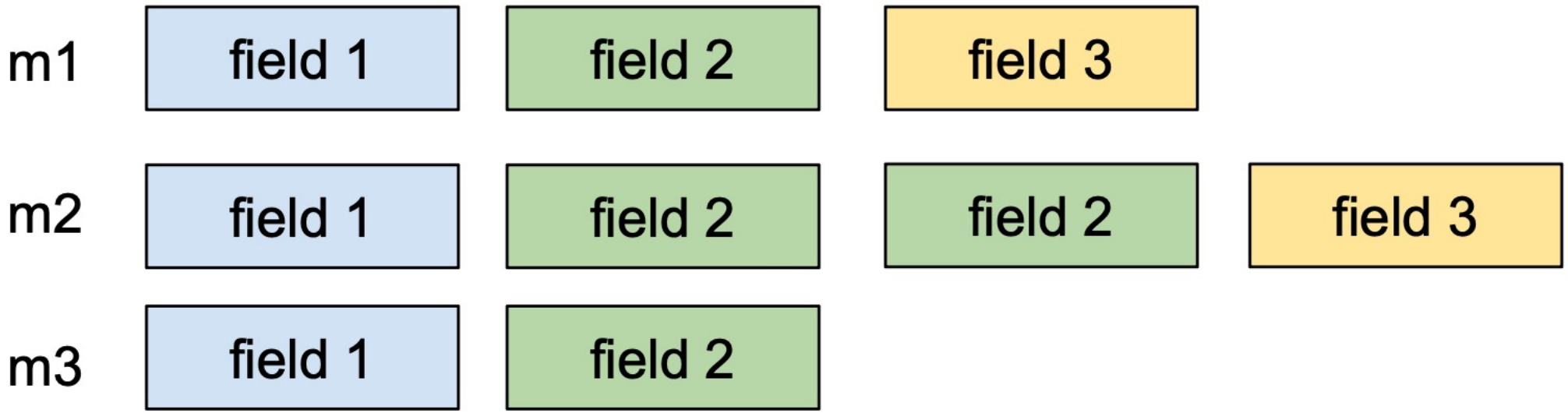
Protobuf 2.0 Arities (Cardinalities)

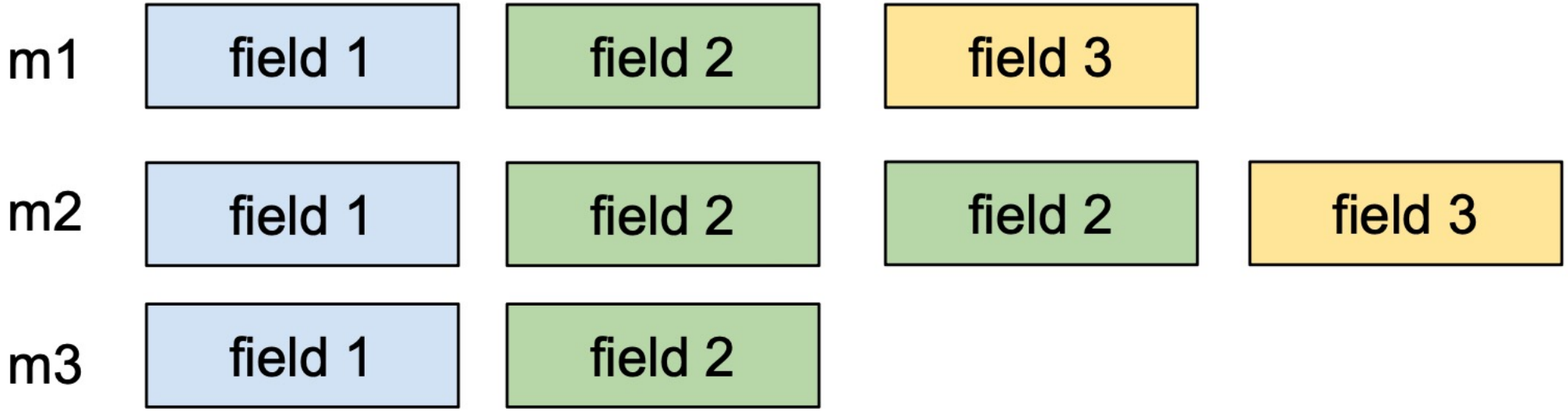
Optional → 0 or 1

Required → 1

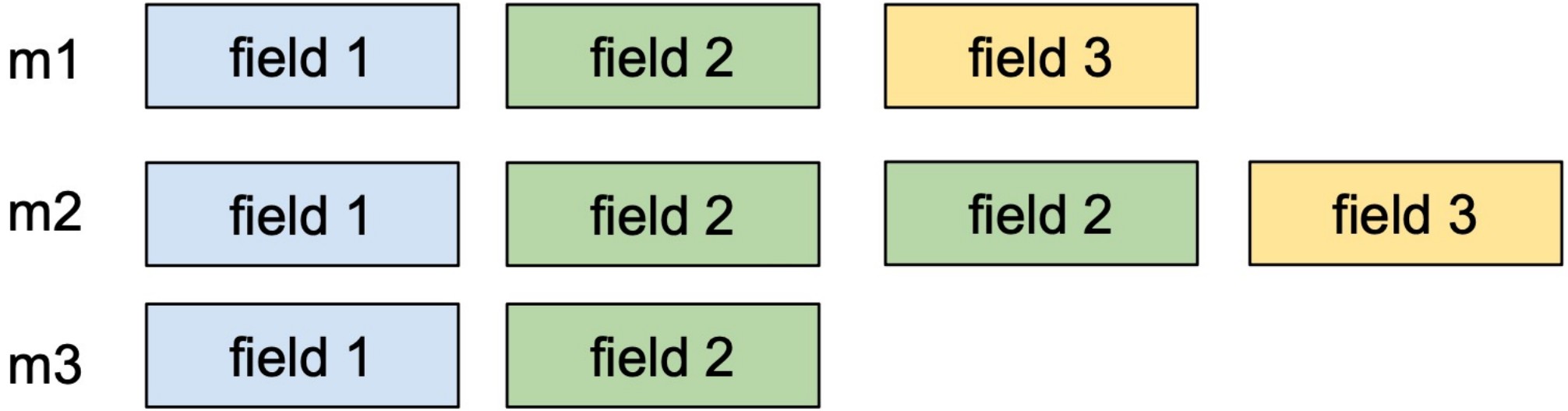
Repeated → 0 or more

Simple Arity Inference Example


m1
field 1
field 2
field 3
m2
field 1
field 2
field 2
field 3
m3
field 1
field 2

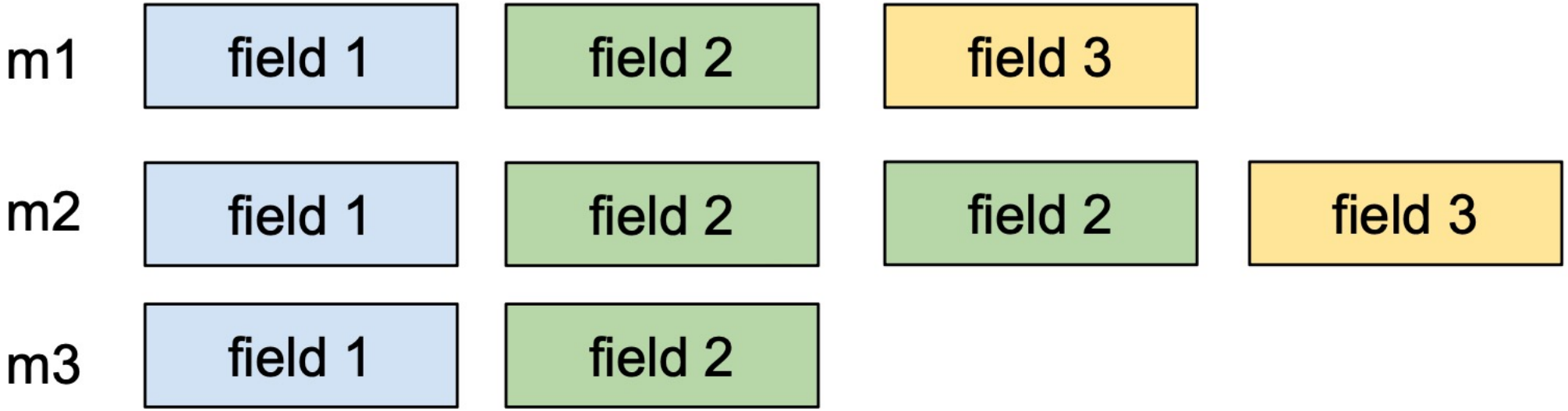


Message	Field1	Field2	Field3
m1	1	1	1
m2	1	2	1
m3	1	1	0



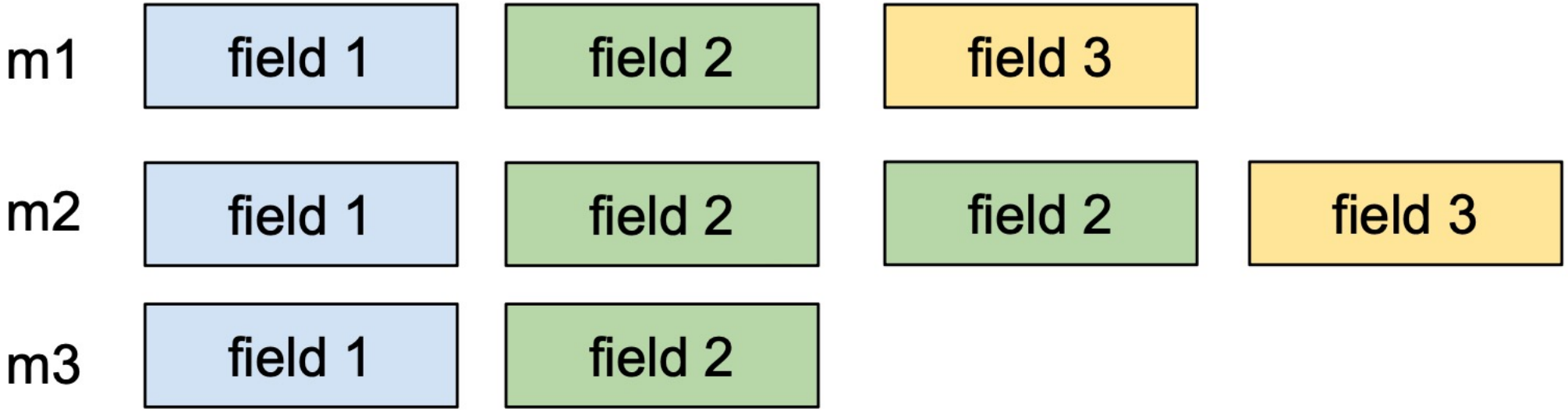
Message	Field1	Field2	Field3
m1	1	1	1
m2	1	2	1
m3	1	1	0

Required



Message	Field1	Field2	Field3
m1	1	1	1
m2	1	2	1
m3	1	1	0

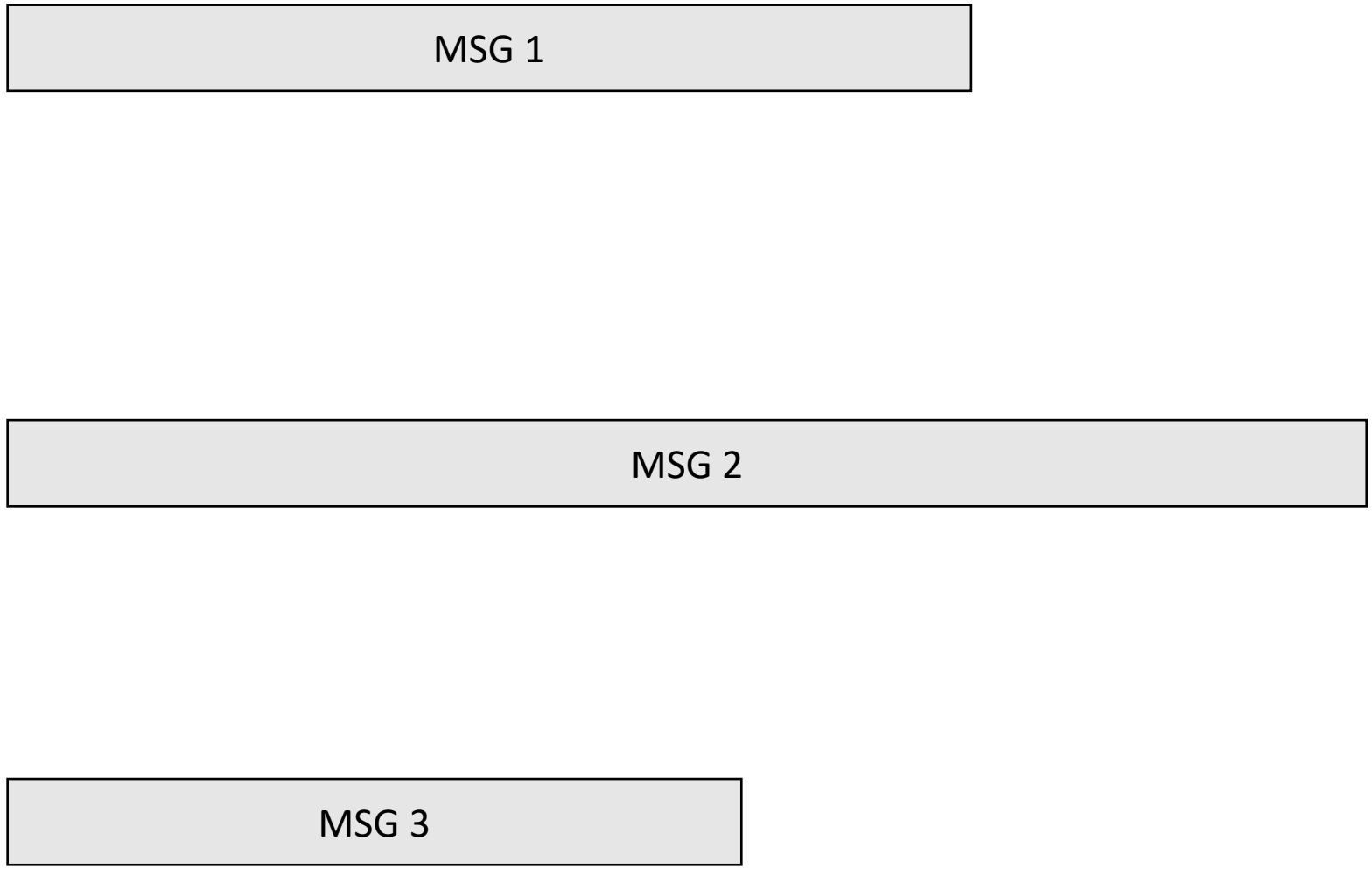
Required **Repeated**

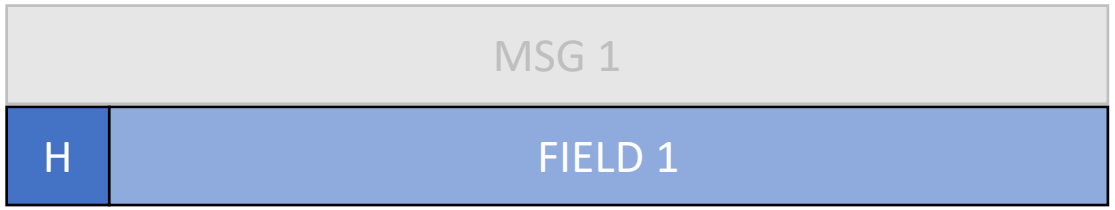


Message	Field1	Field2	Field3
m1	1	1	1
m2	1	2	1
m3	1	1	0

Required **Repeated** **Optional**

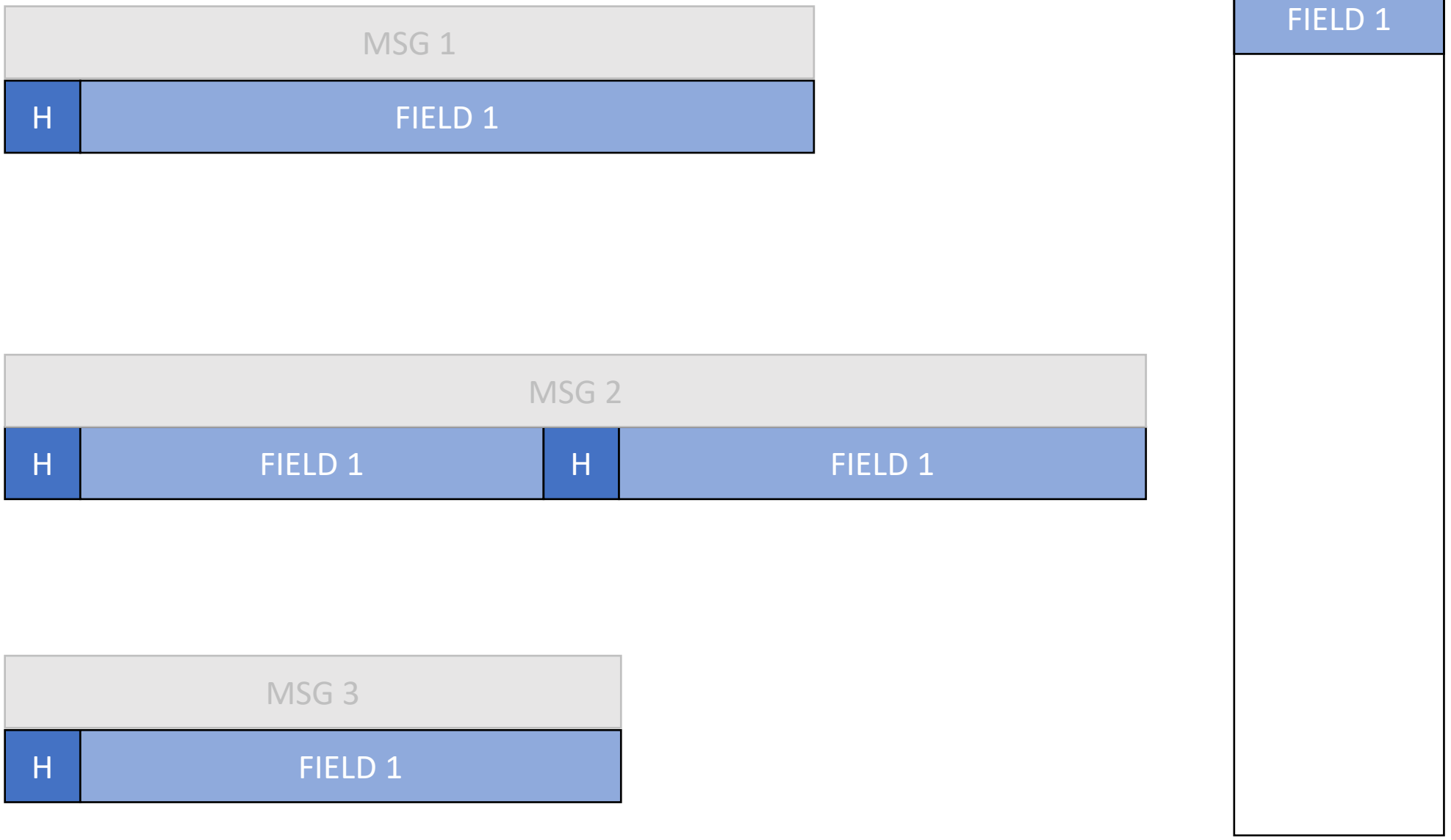
Recursive Inference

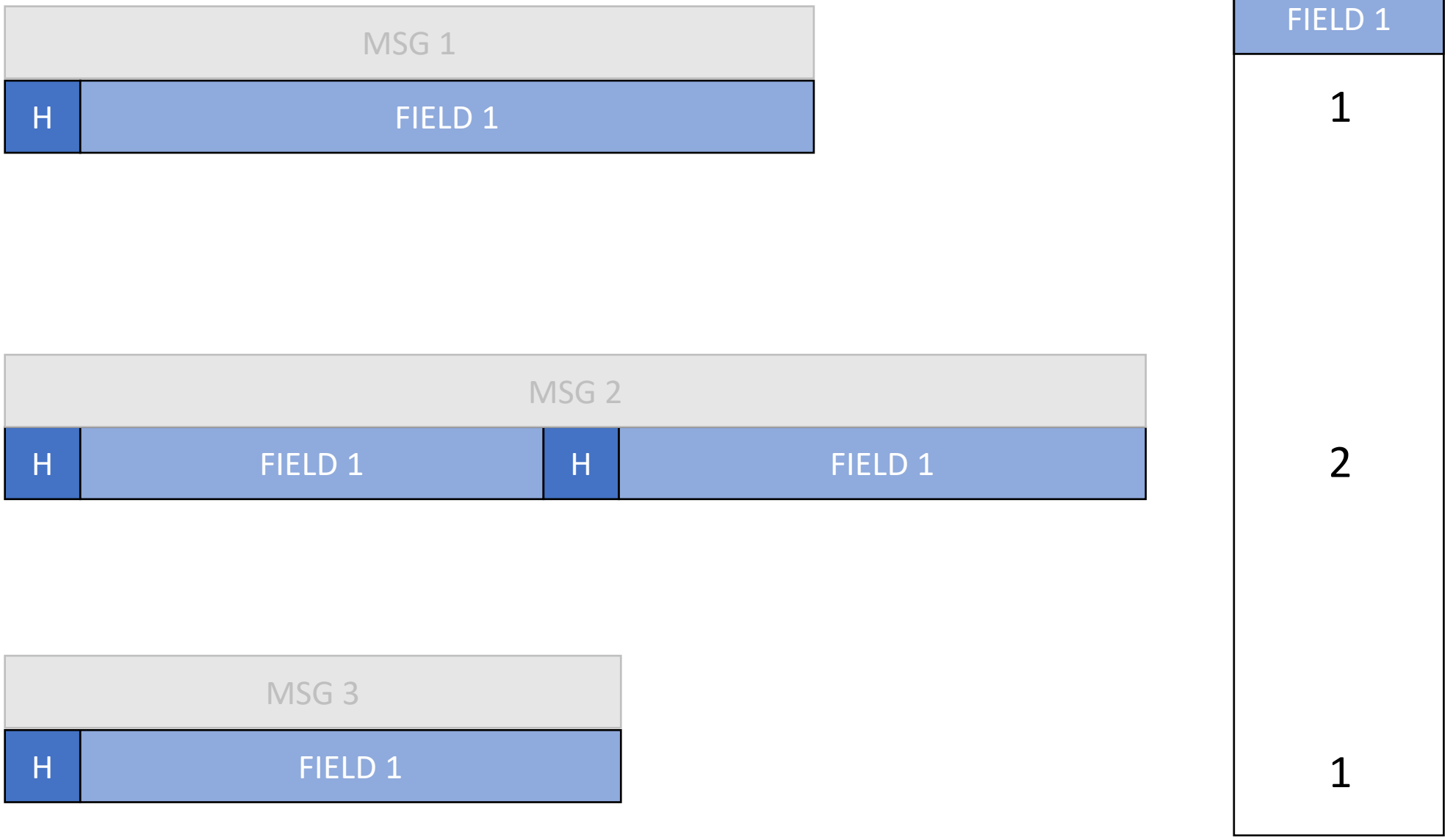

MSG 1
MSG 2
MSG 3

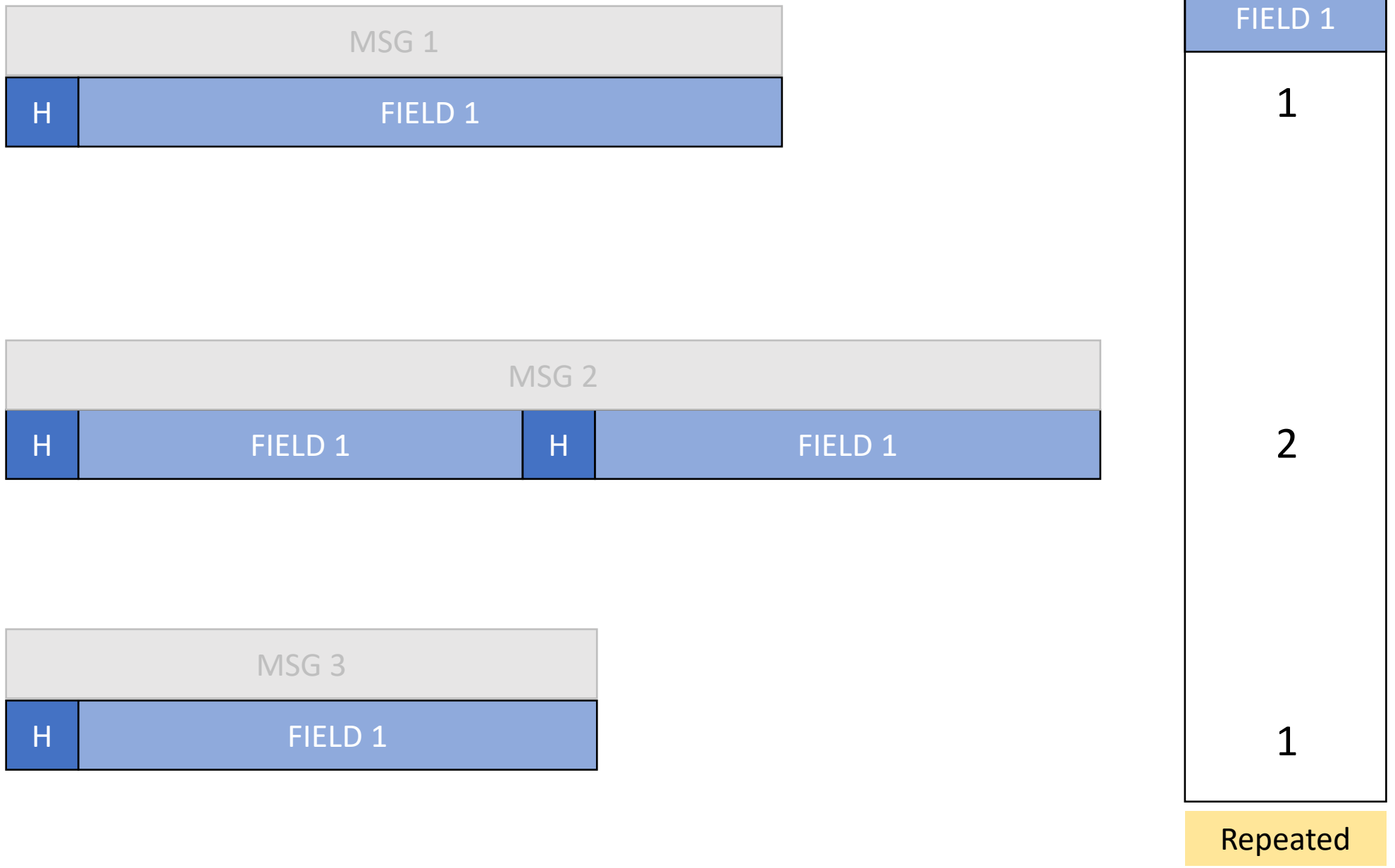

MSG 1
H
FIELD 1

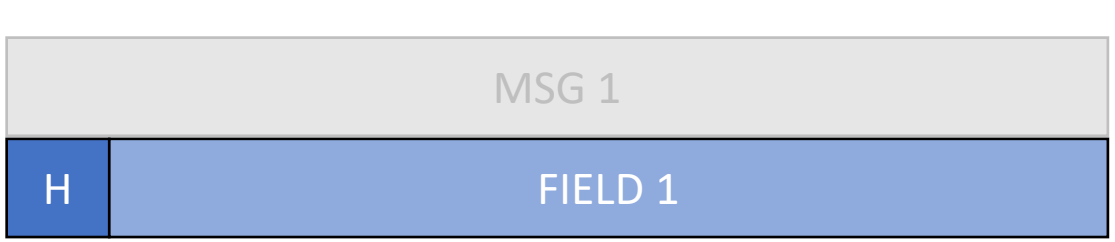

MSG 2
H
FIELD 1
H
FIELD 1

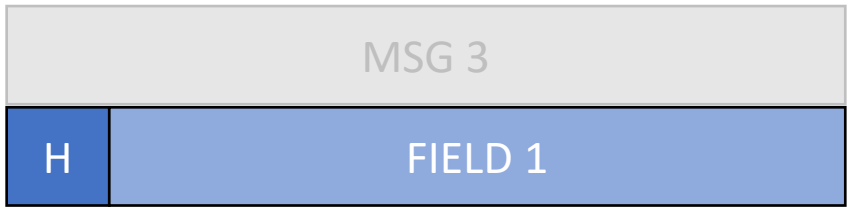

MSG 3
H
FIELD 1

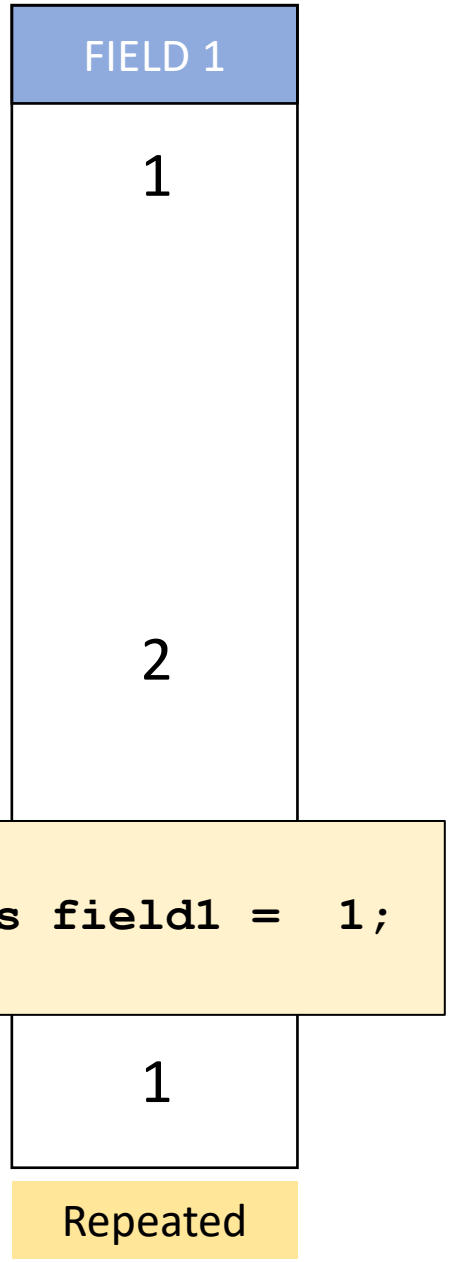

MSG 1
H
FIELD 1
MSG 2
H
FIELD 1
H
FIELD 1
MSG 3
H
FIELD 1
FIELD 1


MSG 1
H
FIELD 1
MSG 2
H
FIELD 1
H
FIELD 1
MSG 3
H
FIELD 1
FIELD 1
1
2
1

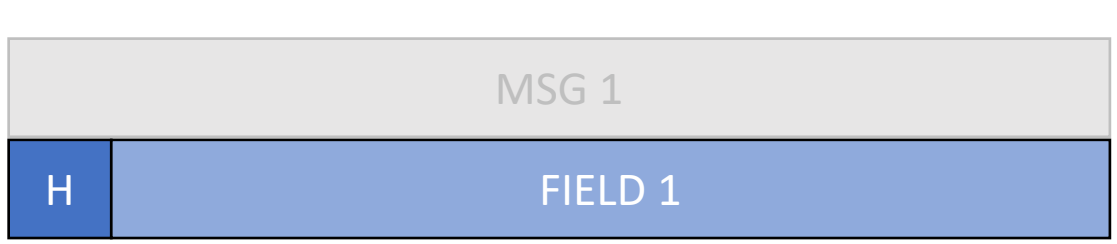

MSG 1
H
FIELD 1
MSG 2
H
FIELD 1
H
FIELD 1
MSG 3
H
FIELD 1
FIELD 1
1
2
1
Repeated

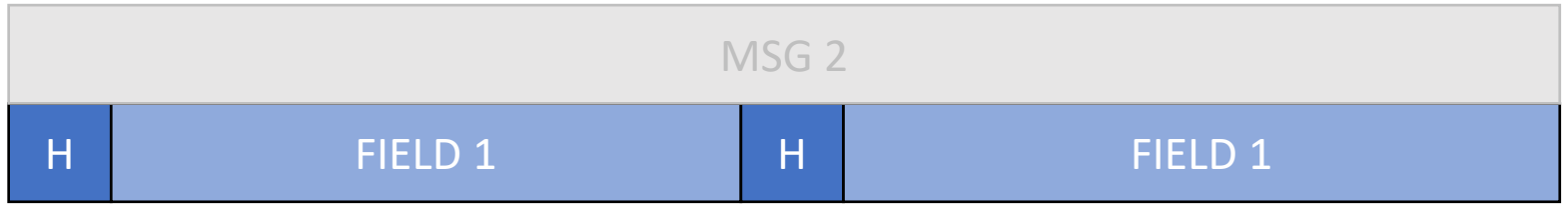


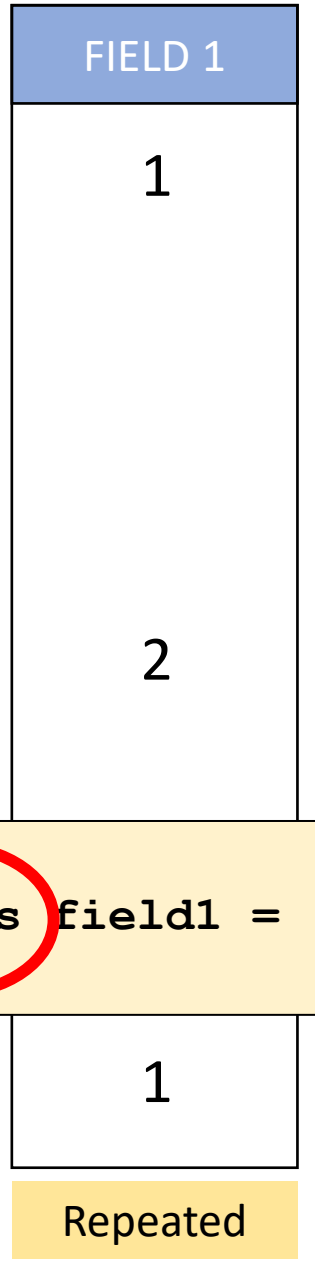




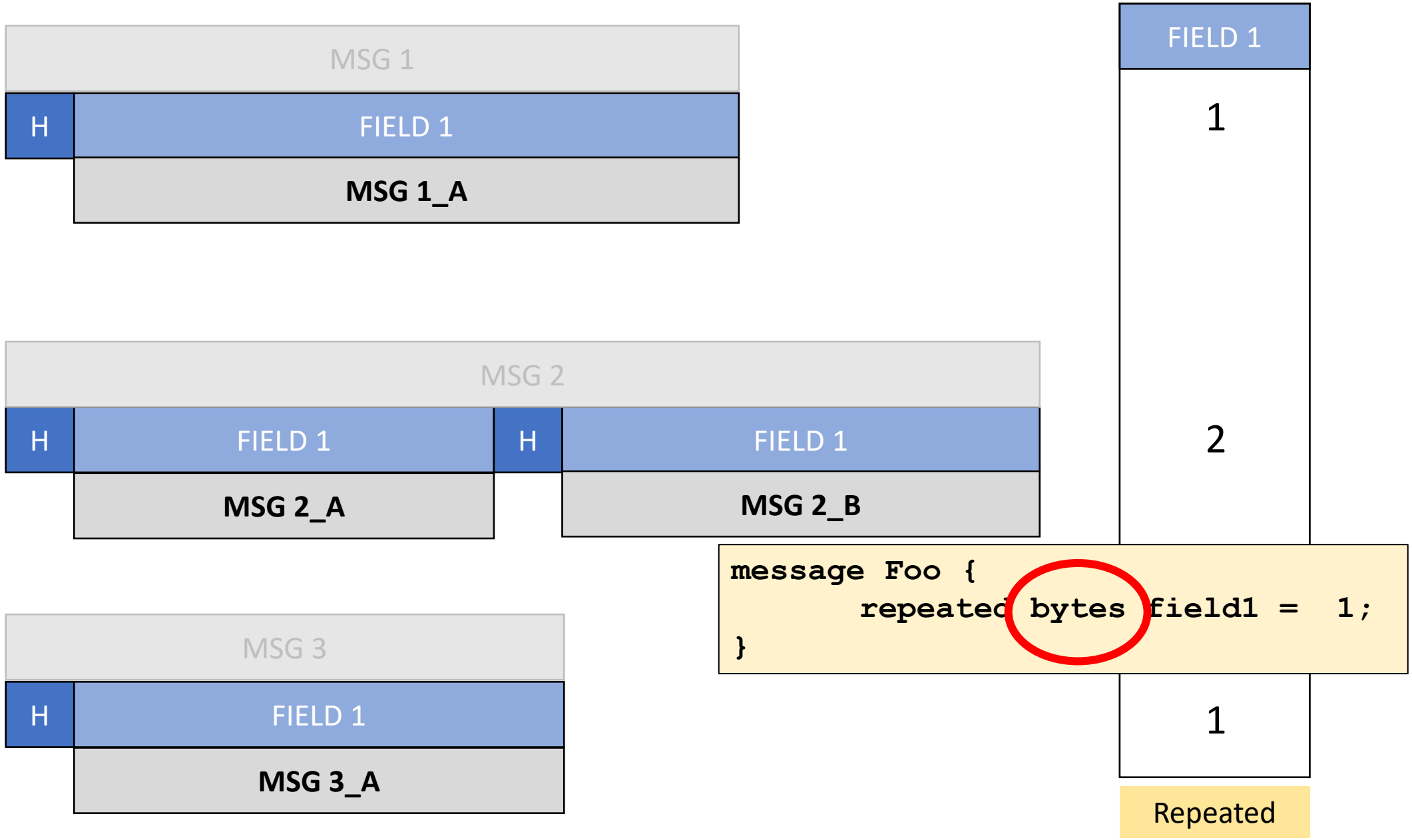

```
message Foo {
        repeated bytes field1 =  1;
}
```

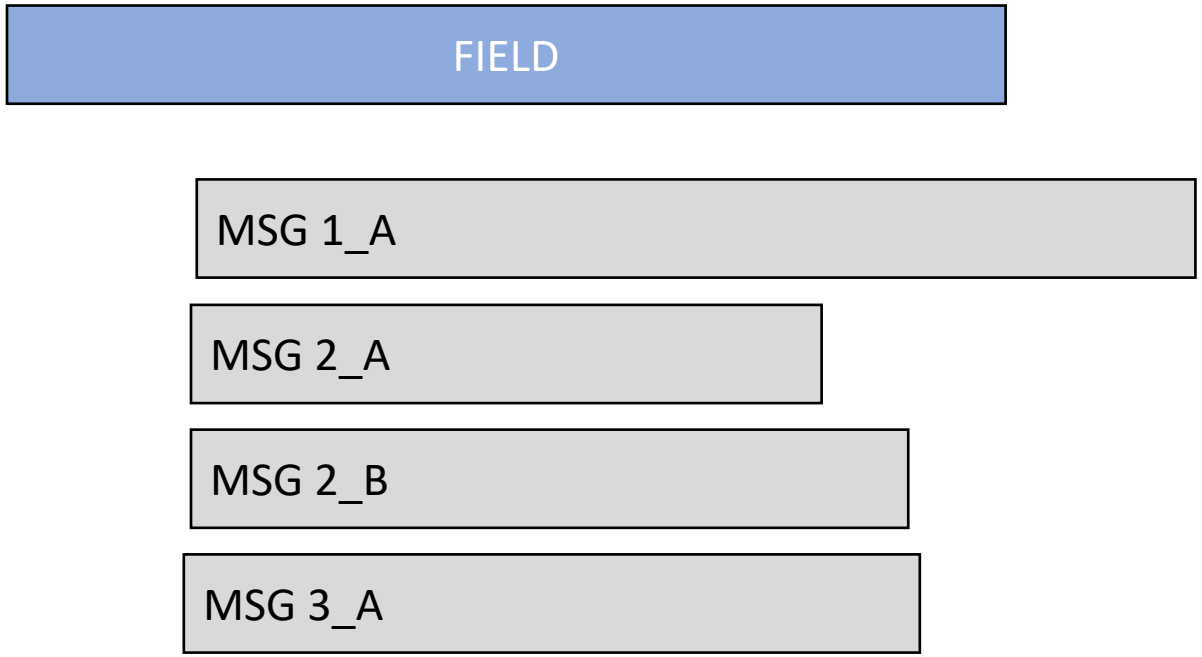







```
message Foo {
        repeated bytes field1 =  1;
}
```


MSG 1
H
FIELD 1
MSG 1_A
MSG 2
H
FIELD 1
MSG 2_A
H
FIELD 1
MSG 2_B
MSG 3
H
FIELD 1
MSG 3_A
FIELD 1
1
2
1
message Foo {
repeated bytes field1 = 1;
}
Repeated


FIELD
MSG 1_A
MSG 2_A
MSG 2_B
MSG 3_A

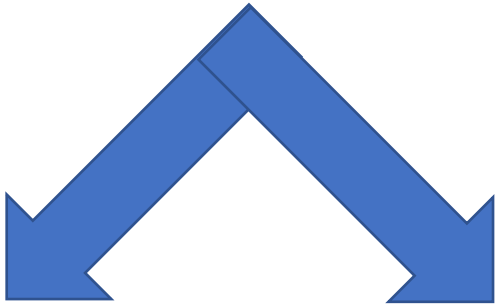
FIELD

MSG 1_A

MSG 2_A

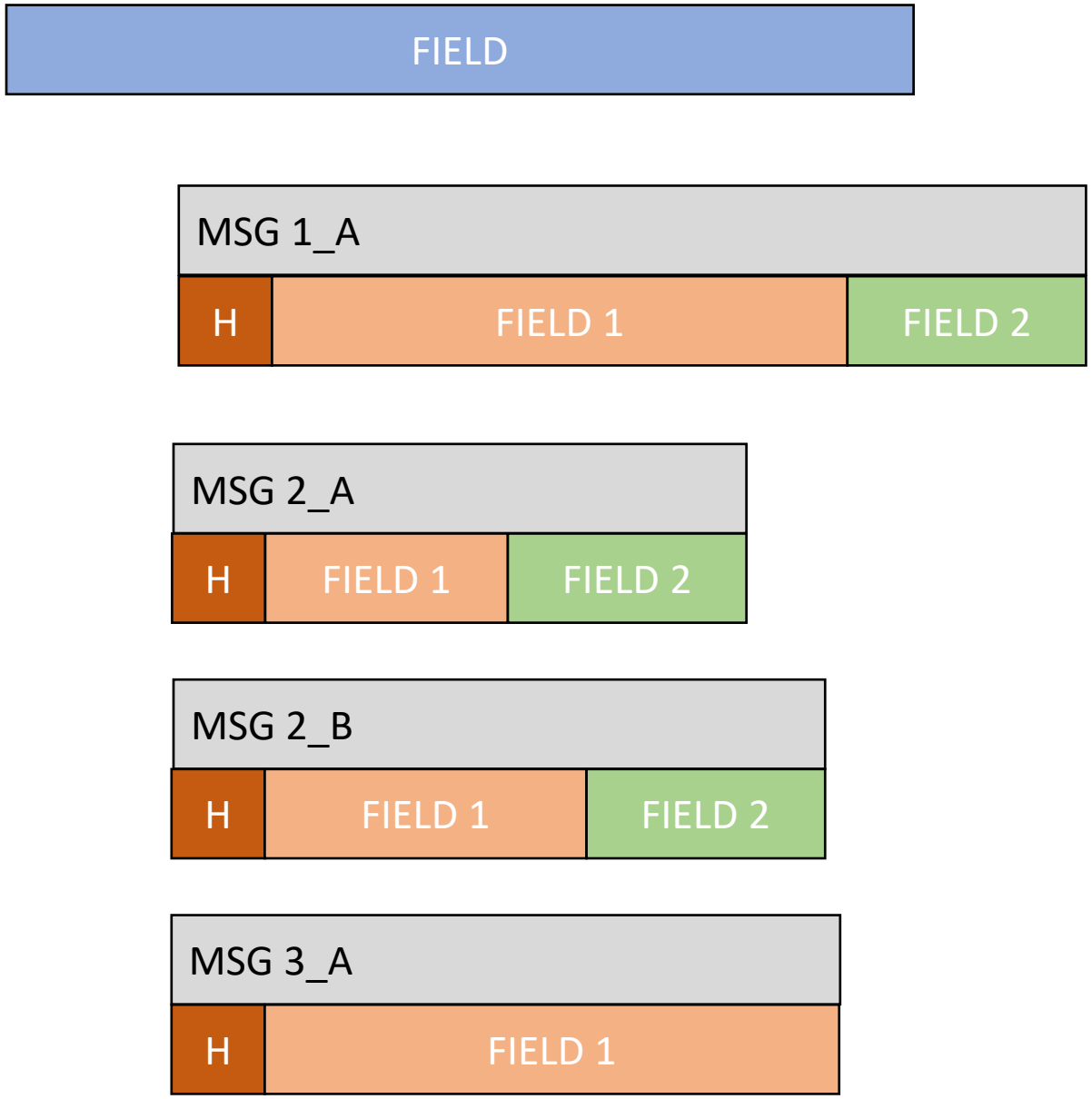
MSG 2_B

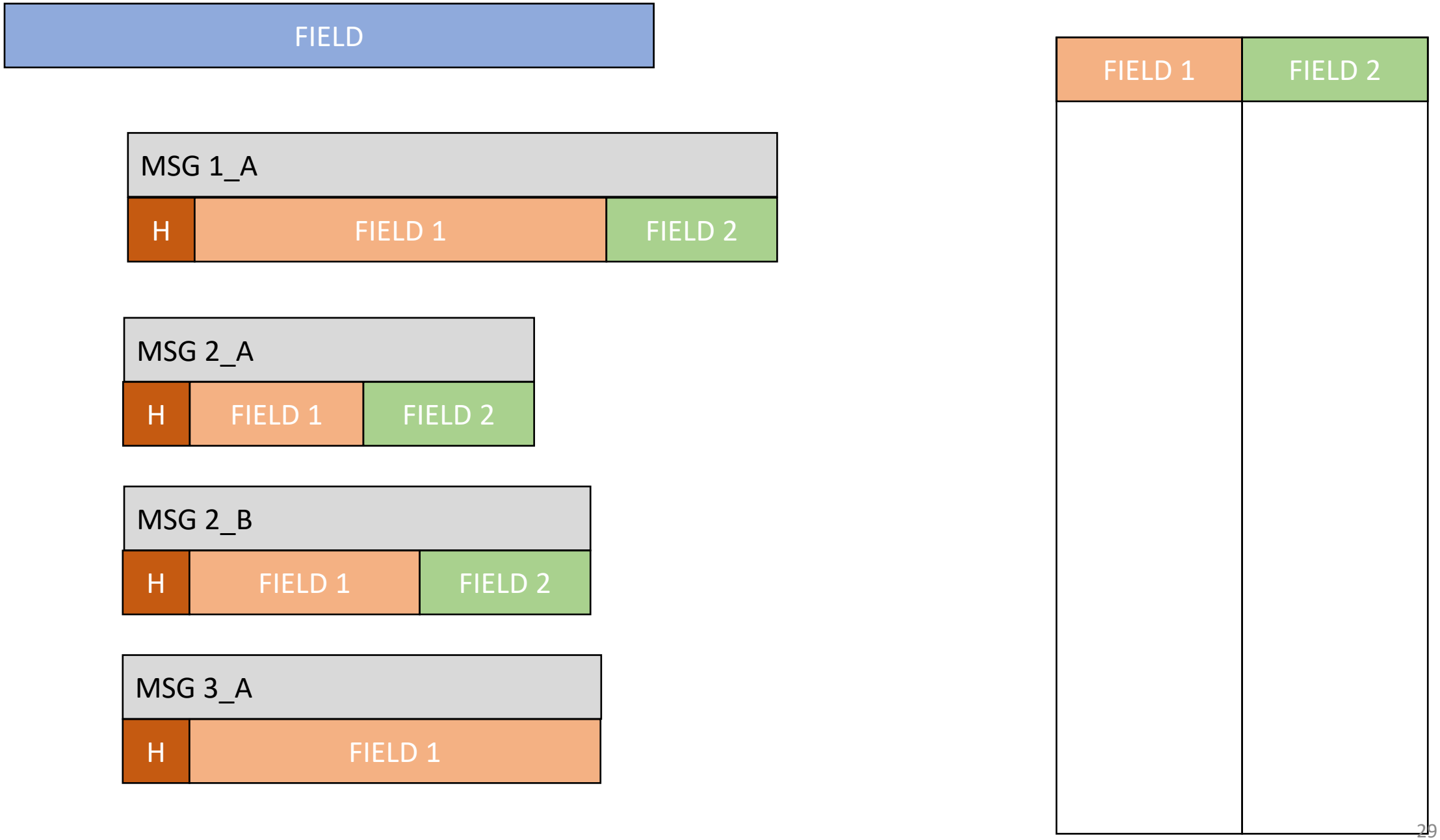
MSG 3_A

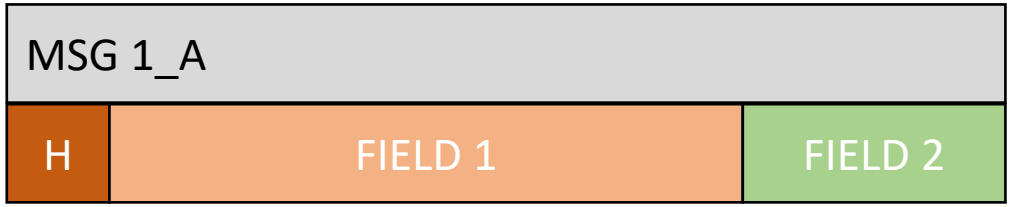


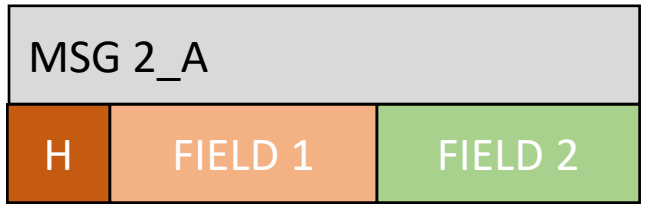
Can’t infer further?
Leave field type as Bytes.

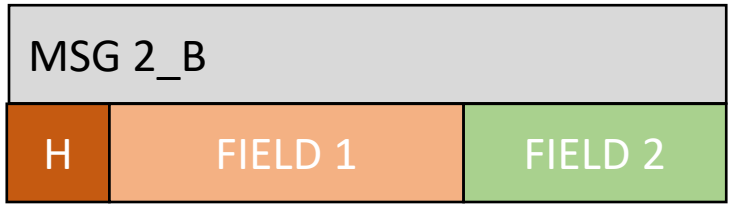
Recursively inferred subformat?
Add subformat to schema.

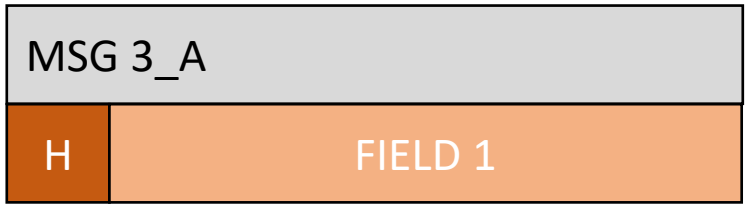

FIELD
MSG 1_A
H
FIELD 1
FIELD 2
MSG 2_A
H
FIELD 1
FIELD 2
MSG 2_B
H
FIELD 1
FIELD 2
MSG 3_A
H
FIELD 1


FIELD
MSG 1_A
H
FIELD 1
FIELD 2
MSG 2_A
H
FIELD 1
FIELD 2
MSG 2_B
H
FIELD 1
FIELD 2
MSG 3_A
H
FIELD 1
FIELD 1
FIELD 2

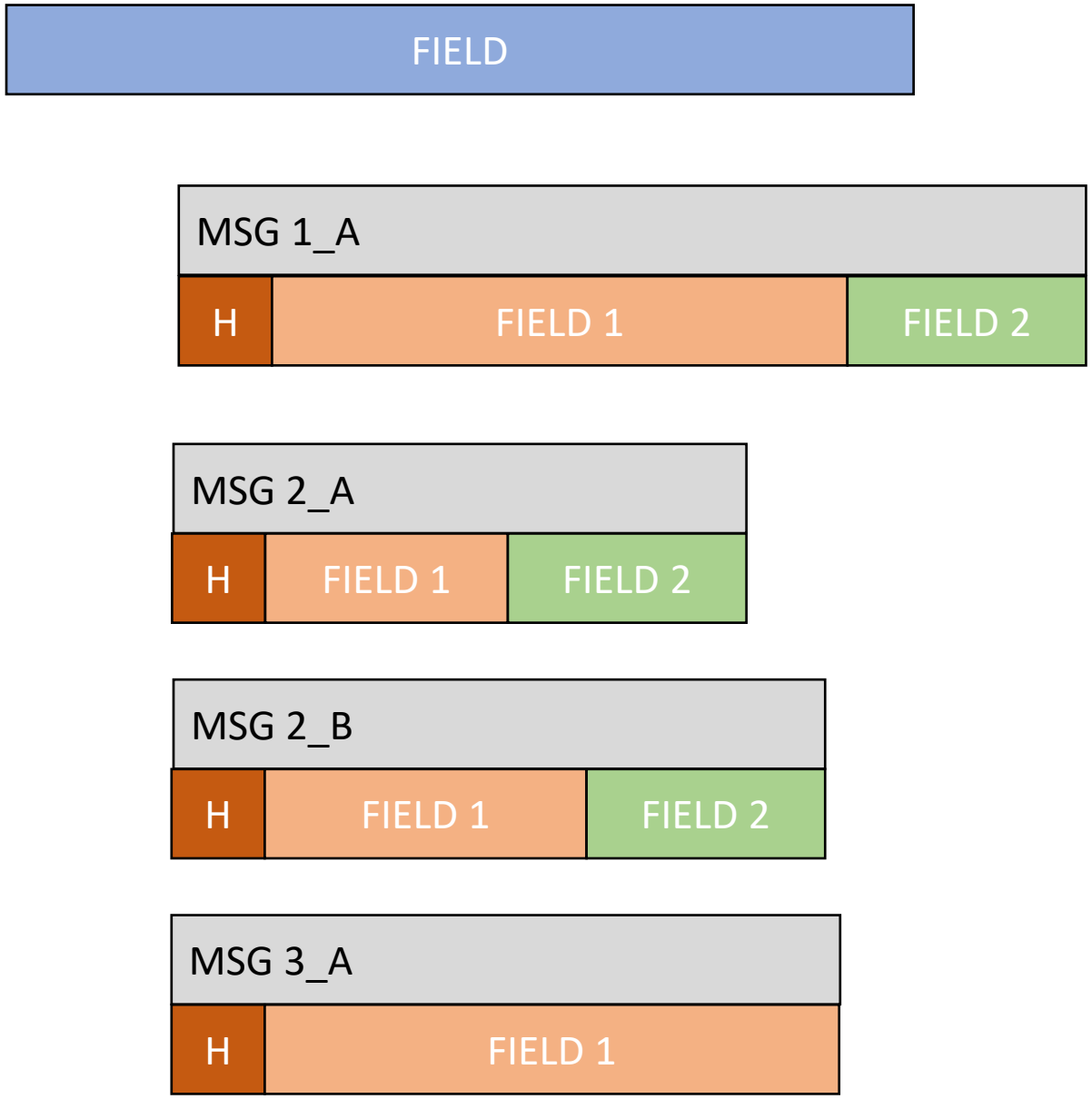




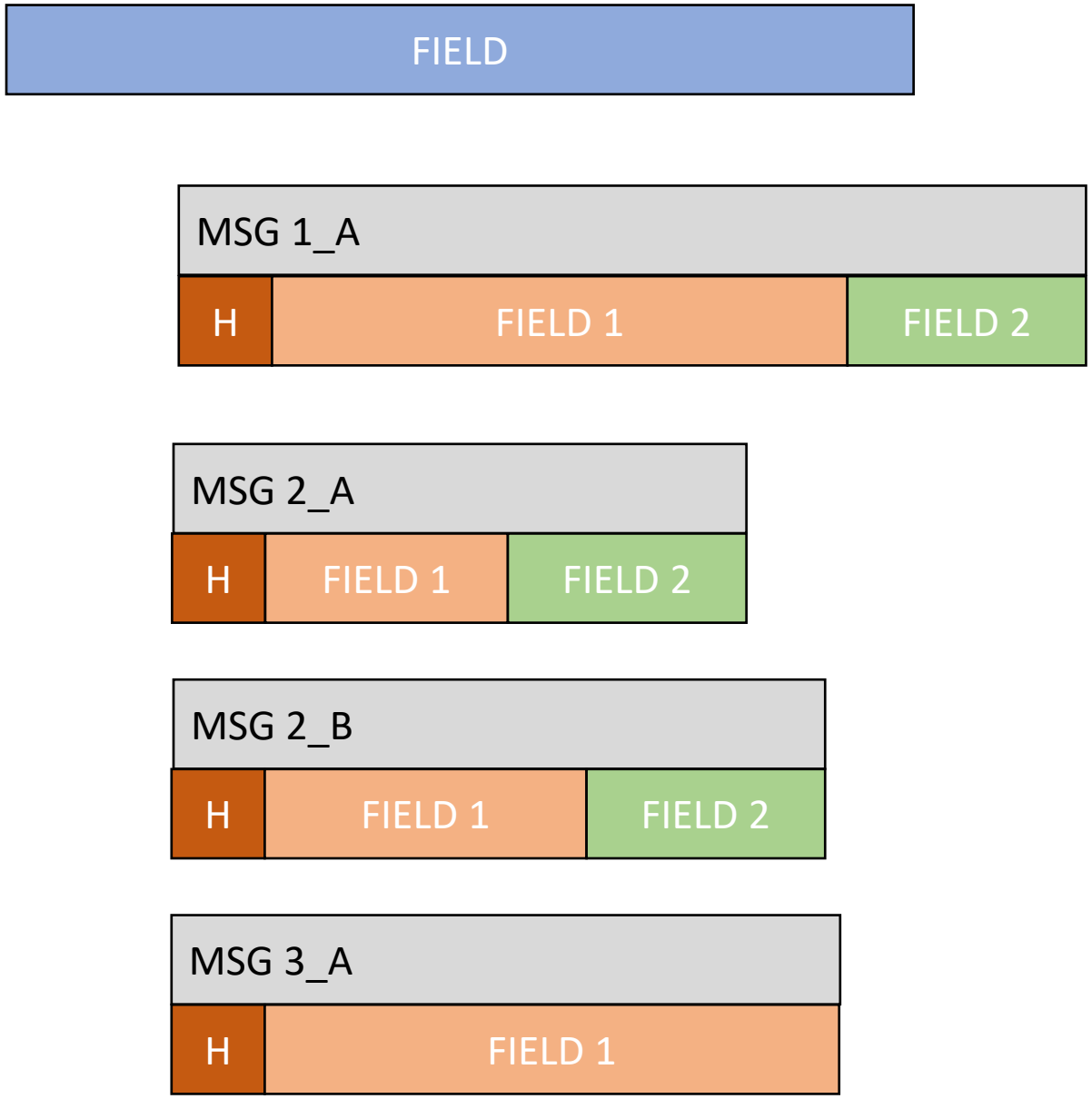




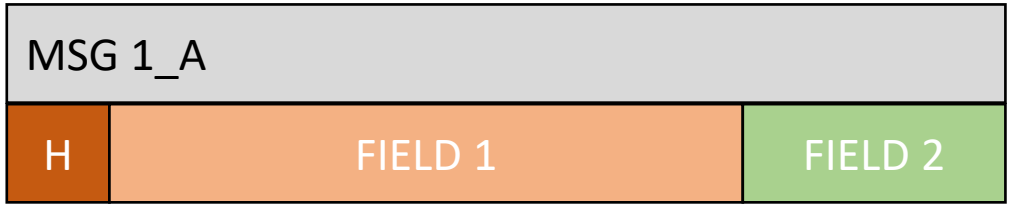
FIELD 1	FIELD 2
1	
1	
1	
1	

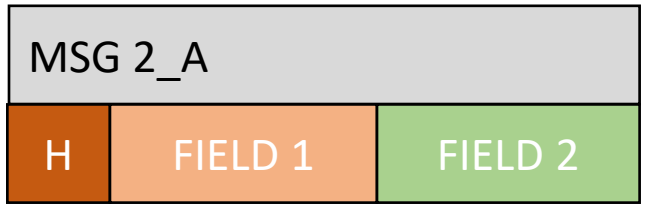


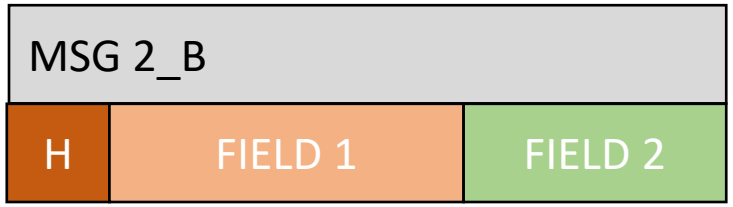
FIELD 1	FIELD 2
1	
1	
1	
1	
Required	

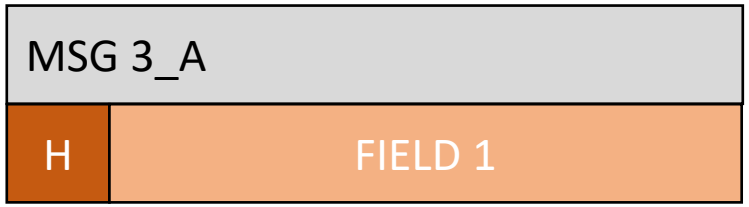


FIELD 1	FIELD 2
1	1
1	1
1	1
1	0
Required	

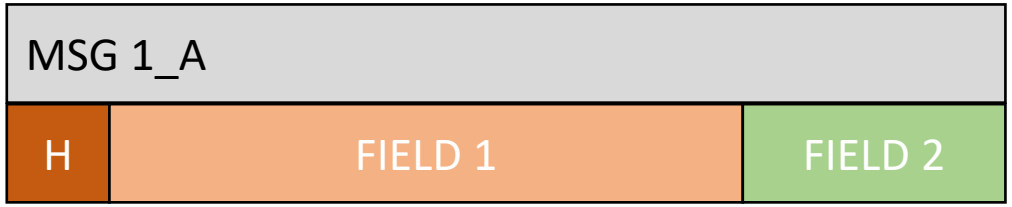


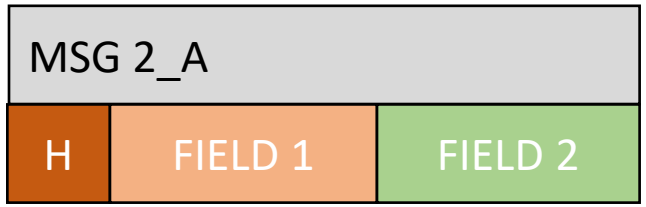


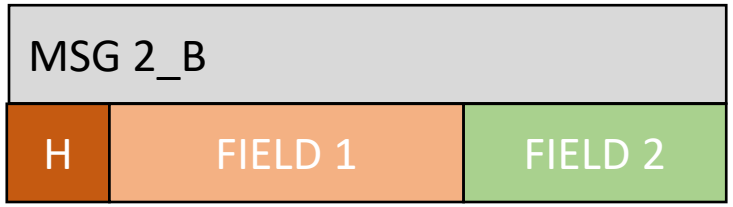


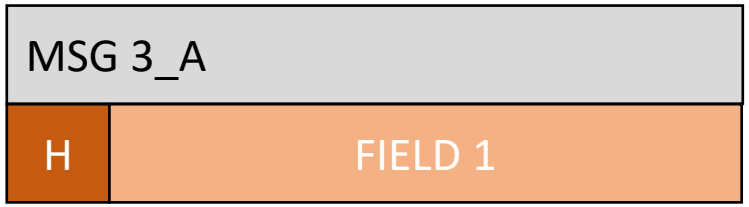


FIELD 1	FIELD 2
1	1
1	1
1	1
1	0
Required	Optional



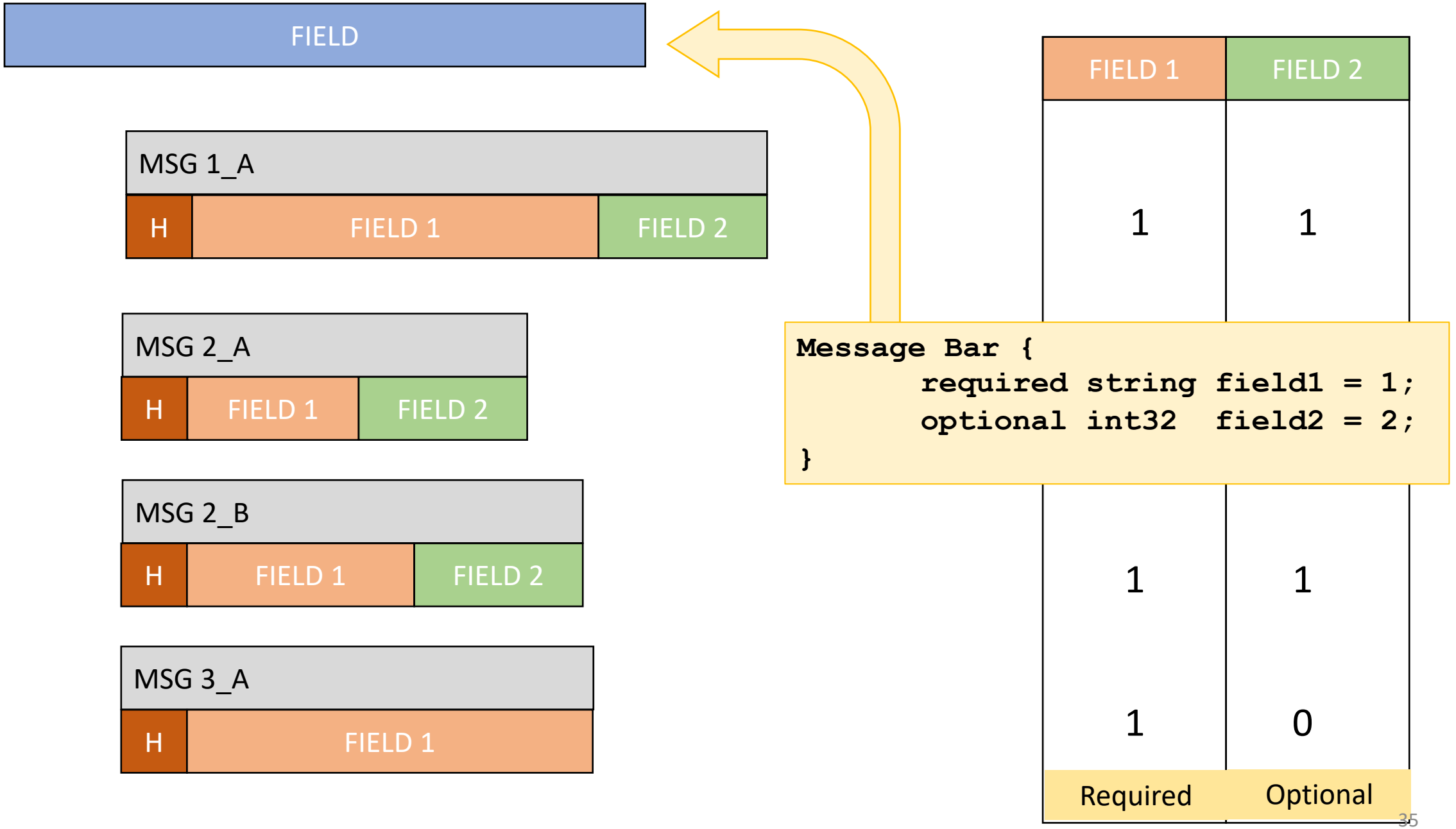


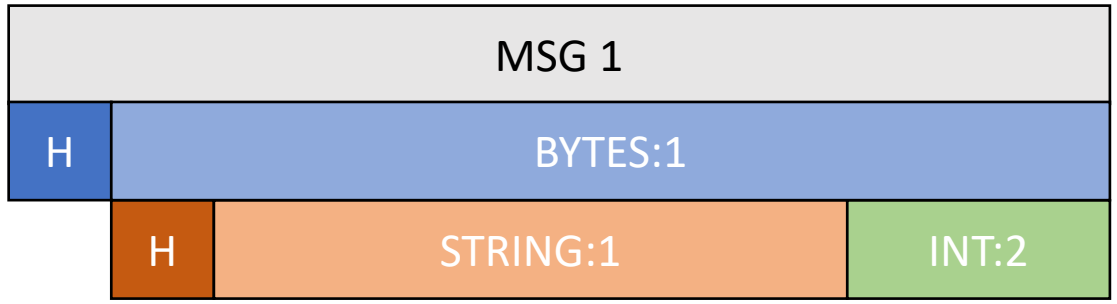





```
Message Bar {
        required string field1 = 1;
        optional int32  field2 = 2;
}
```

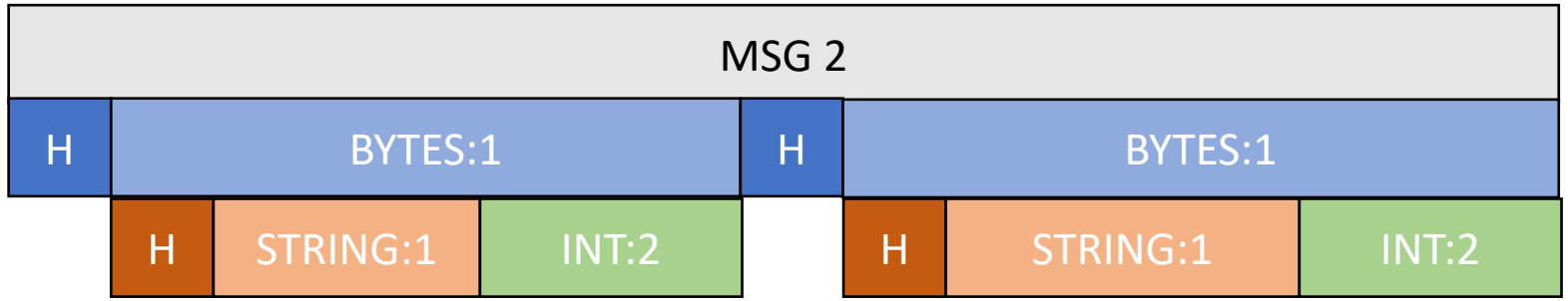
FIELD 1	FIELD 2
1	1
1	1
1	0
Required	Optional

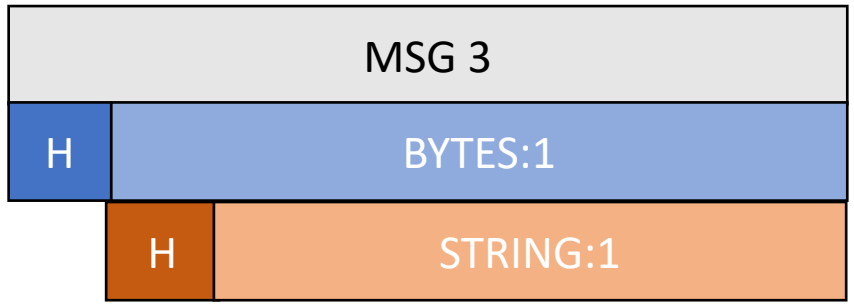

FIELD
MSG 1_A
H
FIELD 1
FIELD 2
MSG 2_A
H
FIELD 1
FIELD 2
MSG 2_B
H
FIELD 1
FIELD 2
MSG 3_A
H
FIELD 1
Message Bar {
required string field1 = 1;
optional int32 field2 = 2;
}
FIELD 1
FIELD 2
1
1
1
1
1
0
Required
Optional




```
message Foo {
        repeated Bar field =  1;
}

Message Bar {
        required string field1 = 1;
        optional int32  field2 = 2;
}
```





Evaluation

- 4 Real world example message formats
- 1 Real world “unknown format”

Evaluation Datasets

• glyph	n=3	10kb – 129kb	glyphs for maps
• grpc	n=2	66 bytes-179 bytes	RCP Network call
• vector	n=5	52kb – 358kb	vector fonts
• mta	n=6	24kb – 128kb	transit data

- mystery file 86Mb

Ground Truth

```
message glyphs {
    repeated fontstack stacks = 1;
}

message fontstack {
    required string name = 1;
    required string range = 2;
    repeated glyph glyphs = 3;
}

message glyph {
    required uint32 id = 1;
    optional bytes bitmap = 2;
    required uint32 width = 3;
    required uint32 height = 4;
    required sint32 left = 5;
    required sint32 top = 6;
    required uint32 advance = 7;
}
```

Ground Truth

```
message glyphs {
    repeated fontstack stacks = 1;
}

message fontstack {
    required string name = 1;
    required string range = 2;
    repeated glyph glyphs = 3;
}

message glyph {
    required uint32 id = 1;
    optional bytes bitmap = 2;
    required uint32 width = 3;
    required uint32 height = 4;
    required sint32 left = 5;
    required sint32 top = 6;
    required uint32 advance = 7;
}
```

Inferred

```
message  Format {
  required Format_A field1 = 1;
}

message  Format_A {
  required bytes    field1 = 1;
  required bytes    field2 = 2;
  repeated Format_B field3 = 3;
}

message  Format_B {
  required int32    field1 = 1;
  optional bytes    field2 = 2;
  required int32    field3 = 3;
  required int32    field4 = 4;
  required int32    field5 = 5;
  required int32    field6 = 6;
  required int32    field7 = 7;
}
```

Ground Truth

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  repeated PhoneNumber phone = 4;
  optional Timestamp last_updated = 5;
  optional bytes portrait_image = 6;
}

message PhoneNumber {
  required string number = 1;
  optional PhoneType type = 2;
}
```

Ground Truth

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  repeated PhoneNumber phone = 4;
  optional Timestamp last_updated = 5;
  optional bytes portrait_image = 6;
}

message PhoneNumber {
  required string number = 1;
  optional PhoneType type = 2;
}
```

Inferred

```
message  Format {
  required bytes    field1 = 1;
  required int32    field2 = 2;
  required bytes    field3 = 3;
  repeated Format_A field4 = 4;
  optional Format_B field5 = 5;
  optional bytes    field6 = 6;
}

message Format_A {
  required bytes    field1 = 1;
  optional int32    field2 = 2;
}

message Format_B {
  required int32    field1 = 1;
}
```

```
message Tile {
  repeated Layer layers = 3;
}

message Layer {
  required string name = 1;
  repeated Feature features = 2;
  repeated string keys = 3;
  repeated Value values = 4;
  optional uint32 extent = 5;
  required uint32 version = 15;
}

message Feature {
  optional uint64 id = 1;
  repeated uint32 tags = 2 [packed=true];
  optional GeomType type = 3;
  repeated uint32 geometry = 4 [packed=true]
}

message Value {
  optional string string_value = 1;
  optional float float_value = 2;
  optional double double_value = 3;
  optional int64 int_value = 4;
  optional uint64 uint_value = 5;
  optional sint64 sint_value = 6;
  optional bool bool_value = 7;
}
```

```
message Tile {
  repeated Layer layers = 3;
}

message Layer {
  required string name = 1;
  repeated Feature features = 2;
  repeated string keys = 3;
  repeated Value values = 4;
  optional uint32 extent = 5;
  required uint32 version = 15;
}

message Feature {
  optional uint64 id = 1;
  repeated uint32 tags = 2 [packed=true];
  optional GeomType type = 3;
  repeated uint32 geometry = 4 [packed=true];
}

message Value {
  optional string string_value = 1;
  optional float float_value = 2;
  optional double double_value = 3;
  optional int64 int_value = 4;
  optional uint64 uint_value = 5;
  optional sint64 sint_value = 6;
  optional bool bool_value = 7;
}
```

```
message  Format {
  repeated Format_A field3 = 3;
}

  message  Format_A {
    required bytes    field1 = 1;
    repeated Format_B field2 = 2;
    repeated bytes    field3 = 3;
    repeated Format_C field4 = 4;
    required int32    field5 = 5;
    required int32    field15 = 1
  }

  message  Format_B {
    required int32    field1 = 1;
    required bytes    field2 = 2;
    required int32    field3 = 3;
    required bytes    field4 = 4;
  }

  message  Format_C {
    optional bytes    field1 = 1;
    // No field 2
    optional int64    field3 = 3;
    optional int32    field4 = 4;
    // No field 5
    // No field 6
    // No field 7
  }
```

```
message Tile {
  repeated Layer layers = 3;
}

message Layer {
  required string name = 1;
  repeated Feature features = 2;
  repeated string keys = 3;
  repeated Value values = 4;
  optional uint32 extent = 5;
  required uint32 version = 15;
}

message Feature {
  optional uint64 id = 1;
  repeated uint32 tags = 2 [packed=true];
  optional GeomType type = 3;
  repeated uint32 geometry = 4 [packed=true];
}

message Value {
  optional string string_value = 1;
  optional float float_value = 2;
  optional double double_value = 3;
  optional int64 int_value = 4;
  optional uint64 uint_value = 5;
  optional sint64 sint_value = 6;
  optional bool bool_value = 7;
}
```

```
message  Format {
  repeated Format_A field3 = 3;
}

  message  Format_A {
    required bytes    field1 = 1;
    repeated Format_B field2 = 2;
    repeated bytes    field3 = 3;
    repeated Format_C field4 = 4;
    required int32    field5 = 5;
    required int32    field15 = 1
  }

  message  Format_B {
    required int32    field1 = 1;
    required bytes    field2 = 2;
    required int32    field3 = 3;
    required bytes    field4 = 4;
  }

  message  Format_C {
    optional bytes    field1 = 1;
    // No field 2
    optional int64    field3 = 3;
    optional int32    field4 = 4;
    // No field 5
    // No field 6
    // No field 7
  }
```

```
message FeedMessage {
  required FeedHeader header = 1;
  repeated FeedEntity entity = 2;
}

message FeedHeader {
  required string gtfs_realtime_ver = 1;
  optional Incr incr = 2;
  optional uint64 timestamp = 3;
  optional string feed_version = 4;
}

message FeedEntity {
  required string id = 1;
  optional bool is_deleted = 2;
  optional TripUpdate trip_update = 3;
  optional VehiclePosition vehicle = 4;
  optional Alert alert = 5;
  optional Shape shape = 6;
  optional Stop stop = 7;
  optional TripMods trip_mods = 8;
}

message TripUpdate {
  required TripDescriptor trip = 1;
  repeated StopTimeUpdate stop_time_update = 2;
  optional VehicleDescriptor vehicle = 3;
  optional uint64 timestamp = 4;
  optional int32 delay = 5;
  optional TripProperties trip_properties = 6;
}

message VehiclePosition {
  optional TripDescriptor trip = 1;
  optional Position position = 2;
  optional uint32 current_stop_sequence = 3;
  optional VehicleStopStatus current_status = 4;
  optional uint64 timestamp = 5;
  optional CongestionLevel congestion_level = 6;
  optional string stop_id = 7;
  optional VehicleDescriptor vehicle = 8;
  optional OccupancyStatus occupancy_status = 9;
  optional uint32 occupancy_percentage = 10;
  repeated CarriageDetails multi_carriage_dets = 11;

}

message Alert {
  repeated TimeRange active_period = 1;
  // No field 2
  // No Field 3
  // No field 4
  repeated EntitySelector informed_entity = 5;
  optional Cause cause = 6;
  optional Effect effect = 7;
  optional TranslatedString url = 8;
  // No Field 9
  optional TranslatedString header_text = 10;
  optional TranslatedString description_text = 11;
  optional TranslatedString tts_header_text = 12;
  optional TranslatedString tts_description_text = 13;
  optional SeverityLevel severity_level = 14;
  optional TranslatedImage image = 15;
  optional TranslatedString image_alternative_text = 16;
  optional TranslatedString cause_detail = 17;
  optional TranslatedString effect_detail = 18;
}
```

```
message  Format {  //FeedMessage
  required Format_A field1 = 1;
  repeated Format_E field2 = 2;
}

message  Format_A { //FeedHeader
  required bytes    field1 = 1;
  // No field 2
  required int32    field3 = 3;
  // No field 4
}

message  Format_E { //FeedEntity
  required bytes    field1 = 1;
  // No field 2
  optional Format_F field3 = 3;
  optional Format_M field4 = 4;
  optional Format_P field5 = 5;
  // No field 6
  // No field 7
  // No Field 8
}

message  Format_F { //TripUpdate
  required Format_G field1 = 1;
  repeated Format_I field2 = 2;
  // No field 3
  // No field 4
  // No Field 5
  // No field 6
}

message  Format_M { //VehiclePosition
  required Format_N field1 = 1;
  // No field 2
  optional int32    field3 = 3;
  optional int32    field4 = 4;
  required int32    field5 = 5;
  // No field 6
  required bytes    field7 = 7;
  // No field 8
  // No Field 9
  // No field 10
  // No Field 11
}

message  Format_P { //Alert
  // No field 1
  // No field 2
  // No Field 3
  // No field 4
  optional Format_Q field5 = 5;
  // No field 6
  // No field 7
  // No field 8
  // No Field 9
  required Format_T field10 = 10;
  // No Field 11
  // No Field 12
  // No Field 13
  // No Field 14
  // No Field 15
  // No Field 16
  // No Field 17
  // No Field 18
}
```

28 ground truth subformats in schema

22 Inferred subformats from observed messages

Mystery File

tausif...@adplay-mobile.com Mar 21, 2019, 5:08:31 AM

to Protocol Buffers

I am new to ProtoBuff file formats. I received a zip file which contains .pb file; the size of the file is approximately 70mb.
I want to decode the file via command line interface.
I have set up protoc environment variable, library from a zip file named this **protoc-3.6.0-win32.zip** downloaded from google protobuf documentation site.

Mystery File

```
message  Format {
  repeated Format_A field1 = 1;
}

message  Format_A {
  repeated bytes    field2 = 2;
  repeated int32    field4 = 4;
  repeated int32    field5 = 5;
  repeated int32    field8 = 8;
  repeated int32    field9 = 9;
  repeated int64    field10 = 10;
  repeated bytes    field11 = 11;
}
```

Mystery File

```
message  Format {
  repeated Format_A field1 = 1;
}

message  Format_A {
  repeated bytes    field2 = 2;
  repeated int32    field4 = 4;
  repeated int32    field5 = 5;
  repeated int32    field8 = 8;
  repeated int32    field9 = 9;
  repeated int64    field10 = 10;
  repeated bytes    field11 = 11;
}
```

Sometimes a string,
Sometimes a subformat
Misusing the field

Results

- Accurately Inferred Schemas with Nested Subformats for Real-World Protobuf samples.
  - Strings vs Bytes
  - Upcasting Integer Sizes
  - Packed Formats
- Having representative samples matter.
- Protobuf can be used for (large) files too!

Future Work & Extensions

- Use Field Values to refine datatypes further.
- Other self-describing formats: flatbuffers, asn1, tlv.
- Names for Fields.

DARTMOUTH

Thank You!

Jared Chandler

jared.d.chandler@dartmouth.edu

This work results from the SPLICE research program, supported by a collaborative award from the National Science Foundation (NSF) SaTC Frontiers program under award number 1955805.