IPM - Understanding State

We are collecting some notes about the state mechanisms in IPM, based on Apache Flink.

- Operator identity, shape, order, parallelism, off/on status.
- Backfilling state from a 3rd source

Statement	Example
All operators **MUST** have a UID. Set by `uid()`.	`dataStream .process(new CountWindowAverage()) .uid("CountWindowAverage") .keyBy("event") .flatMap(new CountWindowAverageMap()) .uid("CountWindowAverageMap") .print().uid("print");`

Operators **CAN** be merged and split	**Before** `dataStream` `.keyBy("event")` `.flatMap(new CountWindowAverageMap())` `.uid("CountWindowAverageMap")` `.print().uid("print");` `dataStream` `.process(new CountWindowAverage())` `.uid("CountWindowAverage")` `.print().uid("print2");` **After** `dataStream` `.process(new CountWindowAverage())` `.uid("CountWindowAverage")` `.keyBy("event")` `.flatMap(new CountWindowAverageMap())` `.uid("CountWindowAverageMap")` `.print().uid("print");`

Operators **CANNOT** share the same UID.	``` dataStream .process(new CountWindowAverage()) .uid("CountWindowAverage") .keyBy("event") .flatMap(new CountWindowAverageMap()) .uid("CountWindowAverageMap") .print().uid("print"); dataStream .process(new CountWindowAverage()) .uid("CountWindowAverage") // <<<<< UID conflict .print().uid("print2"); ```
Operator code body **CAN** change, as long as it has the same UID. If the shape of state changes, the new code body needs to know how to handle the transition from old to new state shape.	You will see this log when the calculated operator body is not matching: `Could not find ExecutionJobVertex. Including user-defined OperatorIDs in search.`
Order of operators **and** the exact shape of input stream **CAN** change as long as UID is kept.	**Before** ``` dataStream .process(new CountWindowAverage()) .uid("CountWindowAverage") .keyBy("event") .flatMap(new CountWindowAverageMap()) .uid("CountWindowAverageMap") .print().uid("print"); ``` **After** ``` dataStream .keyBy("event") .flatMap(new CountWindowAverageMap()) .uid("CountWindowAverageMap") .process(new CountWindowAverage()) .uid("CountWindowAverage") .print().uid("print"); ```

An operator **CANNOT** be disabled and re-enabled with a savepoint taken when it was disabled.	
An operator **CAN** have different parallelism in a later deployment and still use the same savepoint from a previous point.	**Before** `dataStream .keyBy("event") .flatMap(new CountWindowAverageMap()) .uid("CountWindowAverageMap") .process(new CountWindowAverage()) .uid("CountWindowAverage") .print().uid("print");` **After** `dataStream .keyBy("event") .flatMap(new CountWindowAverageMap()).setParallelism(2) .uid("CountWindowAverageMap") .process(new CountWindowAverage()).setParallelism(1) .uid("CountWindowAverage") .print().uid("print");`
Operators **CAN** be started from different entry points and still use/preserve their states.	**Before** `./bin/standalone-job.sh start-foreground --job-classname com.motaword.ipm.kernel.Application` **After** `./bin/standalone-job.sh start-foreground --job-classname com.motaword.ipm.kernel.ApplicationNew`