Software for Satellites

Notes about software development
for scientific space missions

Data (definitions inspired to ASN.1)

PER: Packet Encoding Rule
Something to transmit along a line

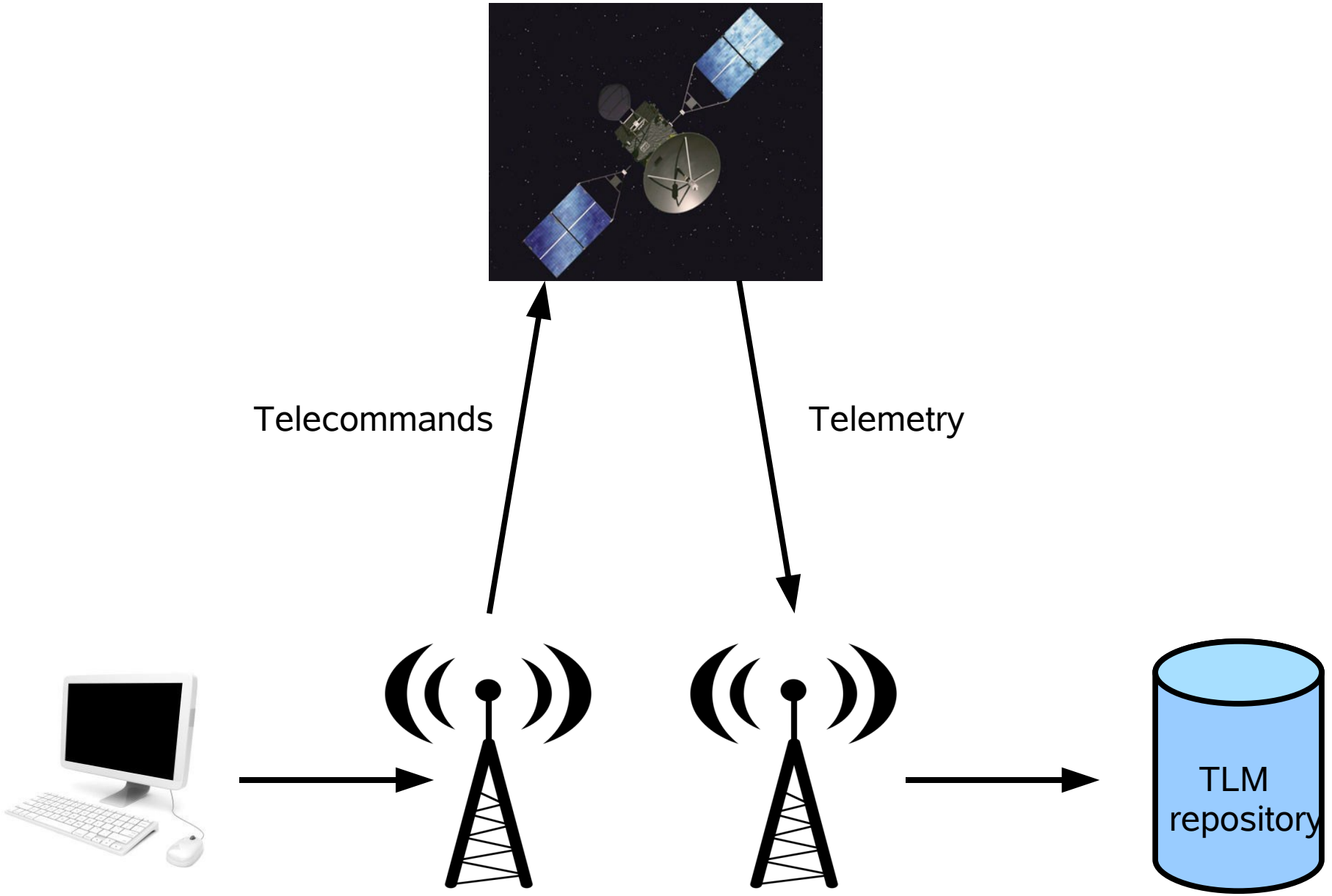
MER: Memory Encoding Rule
The working format for a program

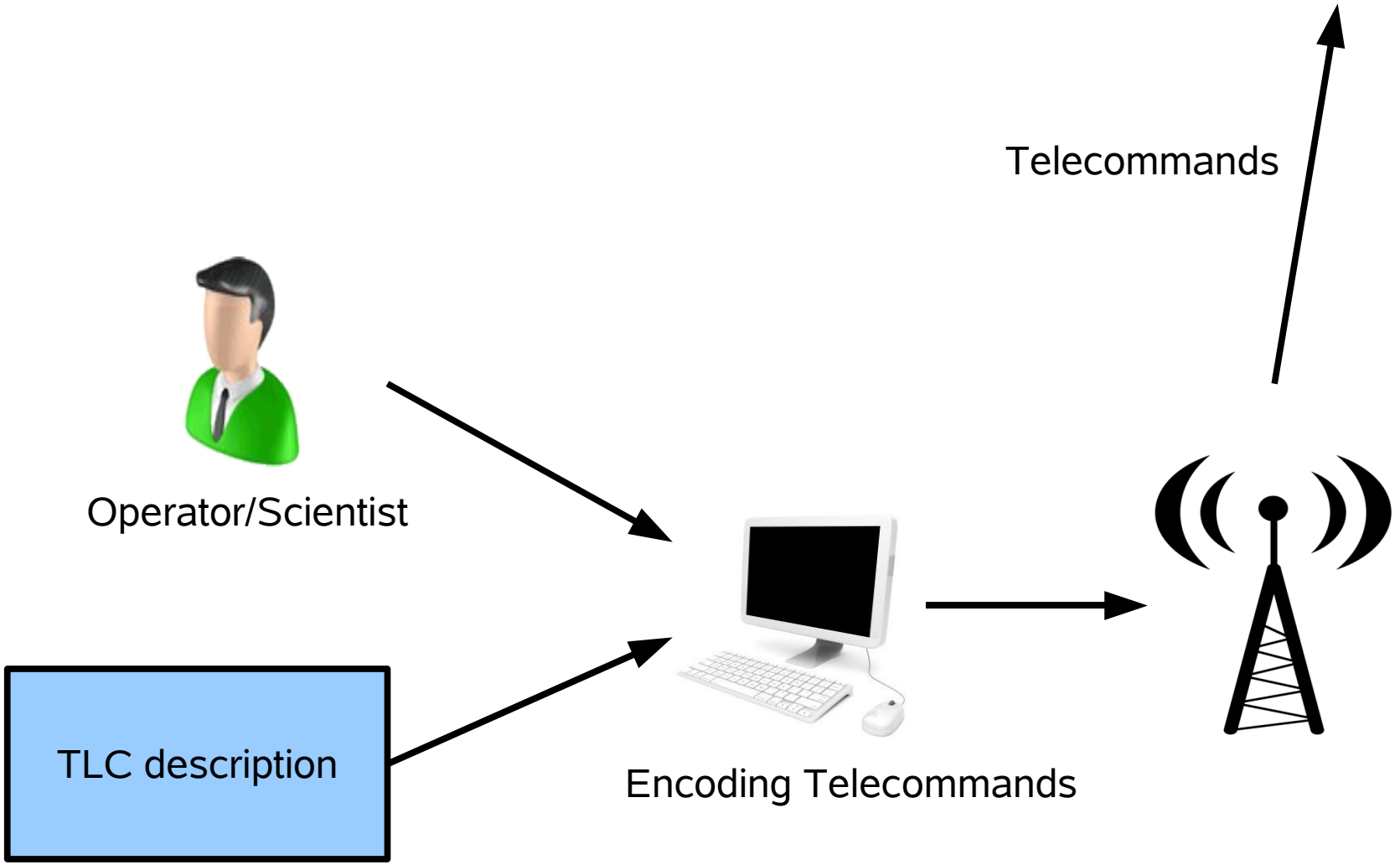
HER: Human Encoding Rule
Something readable for humans

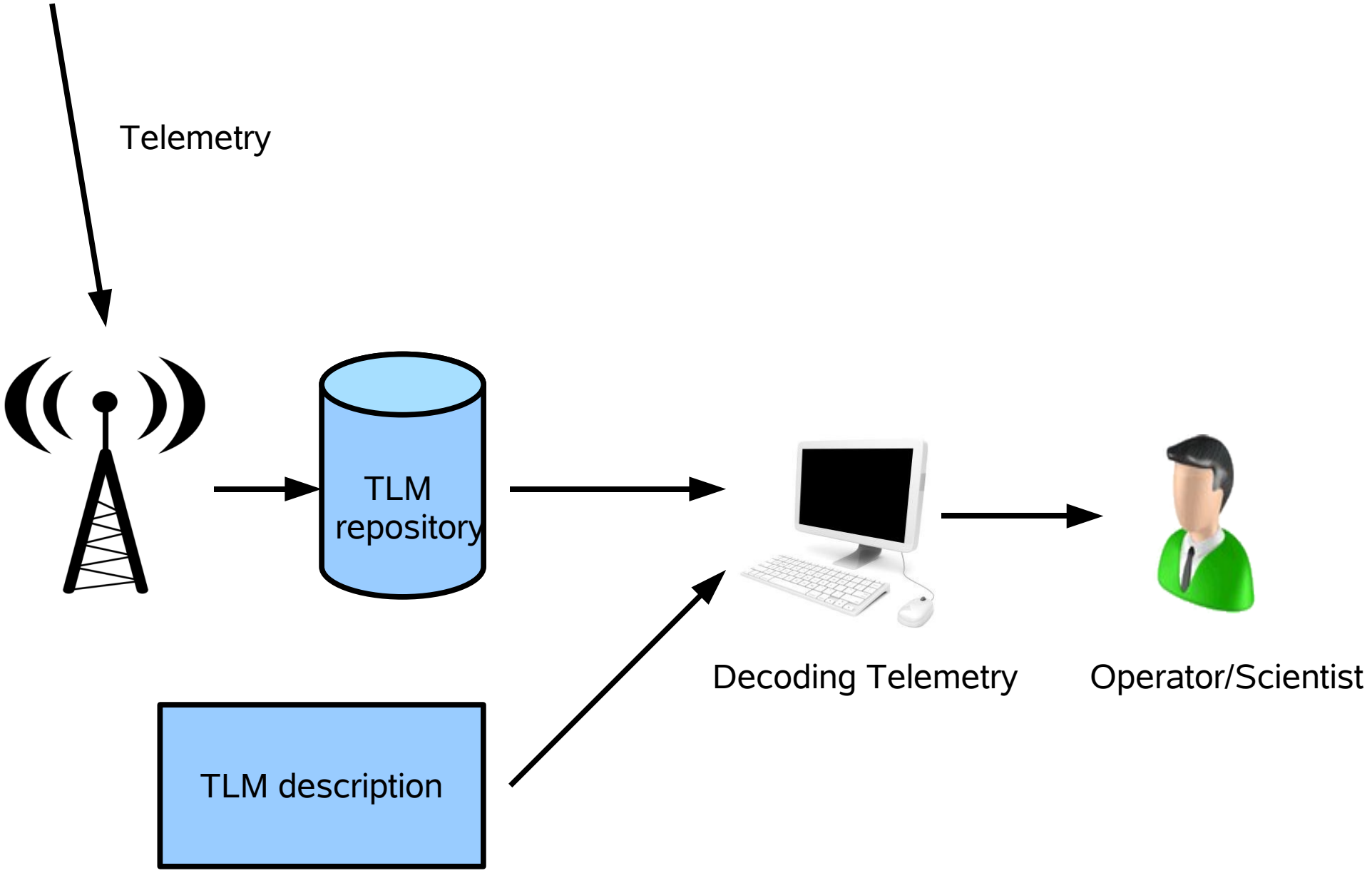
Algorithms

An Algorithm is a set of rules that precisely defines a sequence of operations

There is a strong mutual dependencency between Data Structures and the Algorithms handling them


Telecommands
Telemetry
TLM
repository

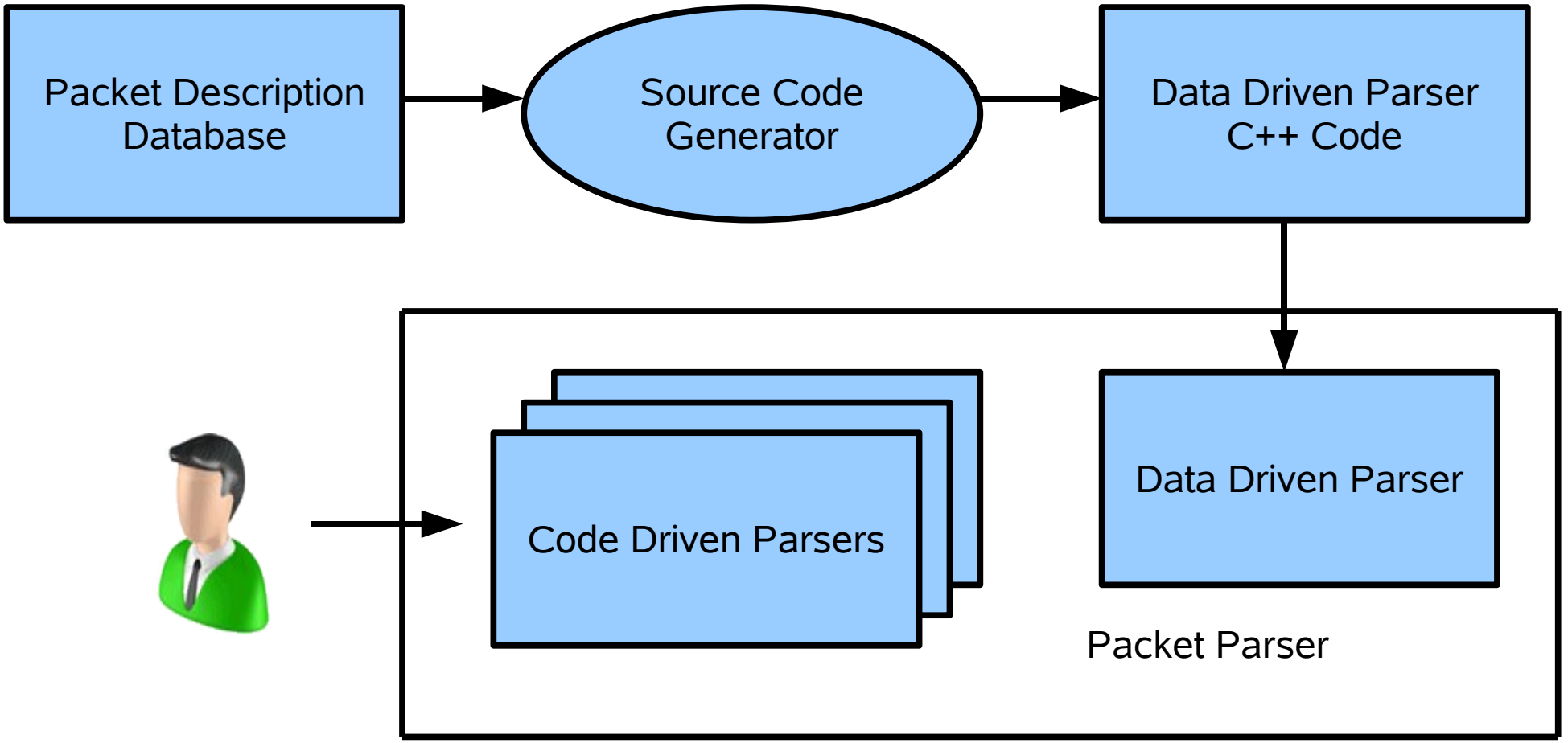

Telecommands
Operator/Scientist
TLC description
Encoding Telecommands

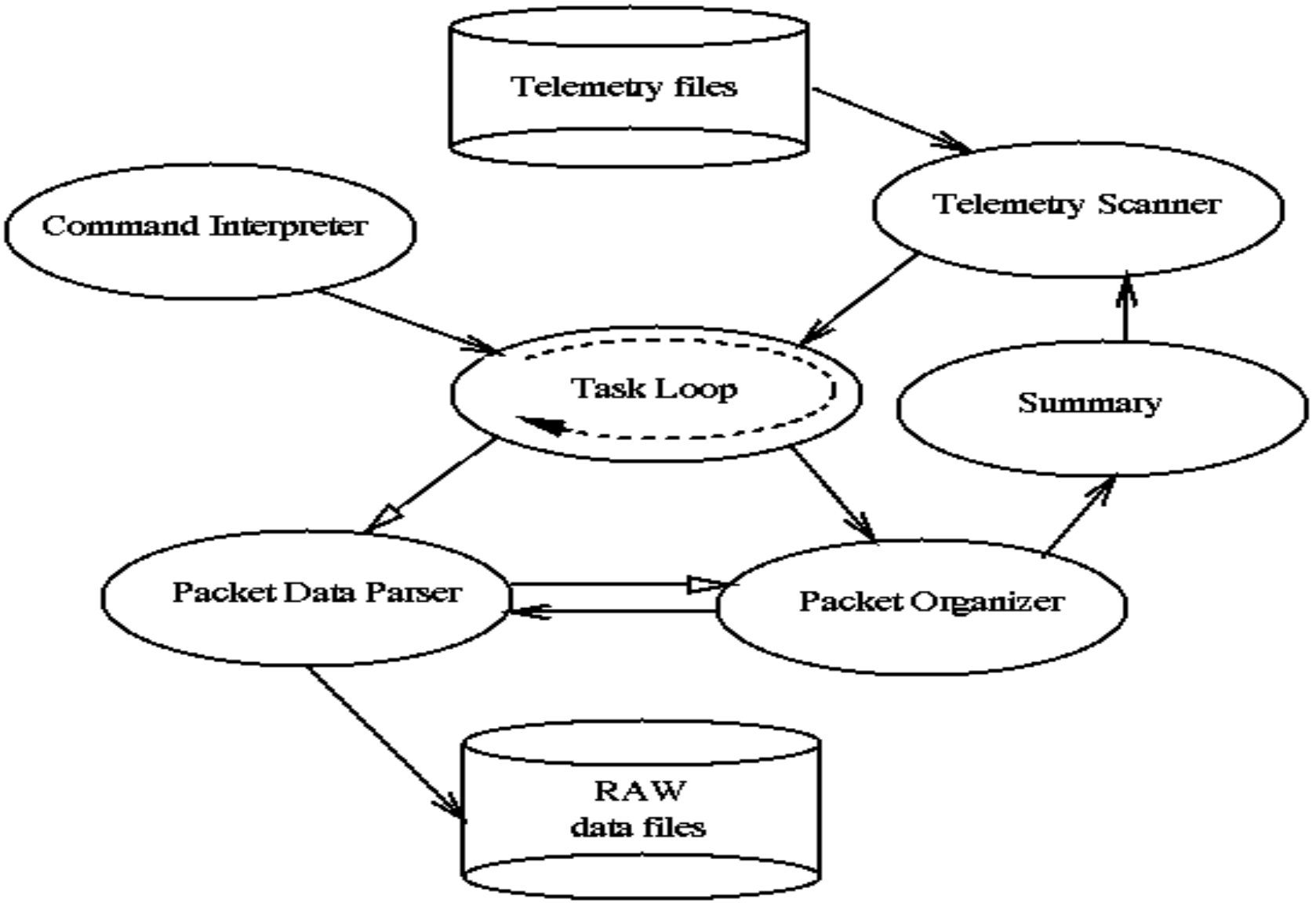

Telemetry
TLM repository
TLM description
Decoding Telemetry
Operator/Scientist

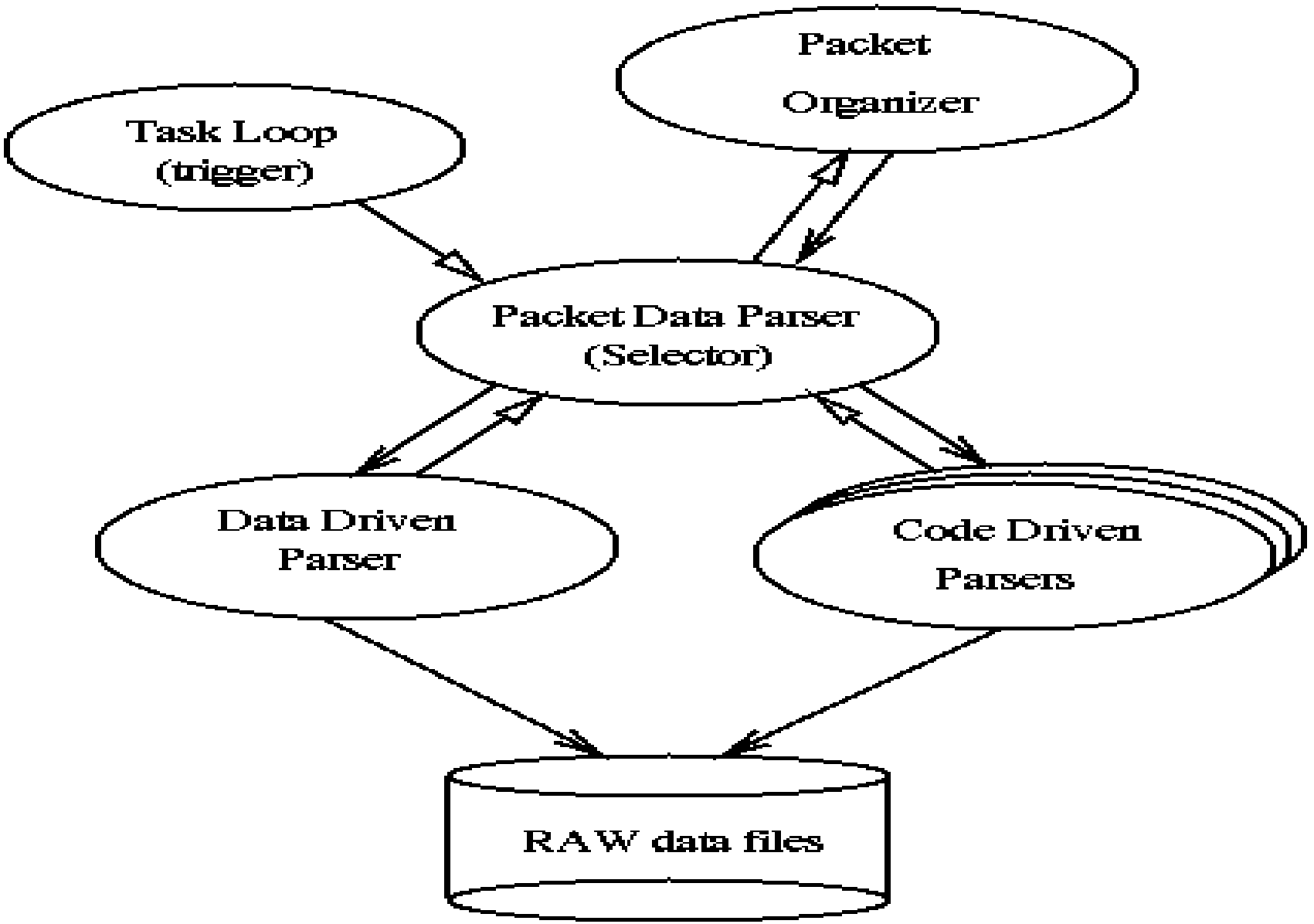
Example 1

How metadata may be used to automatically generate software routines for parsing or formatting satellite packets

INTEGRAL Preprocessing
Packet Parser




Telemetry files
Telemetry Scanner
Command Interpreter
Task Loop
Summary
Packet Data Parser
Packet Organizer
RAW
data files

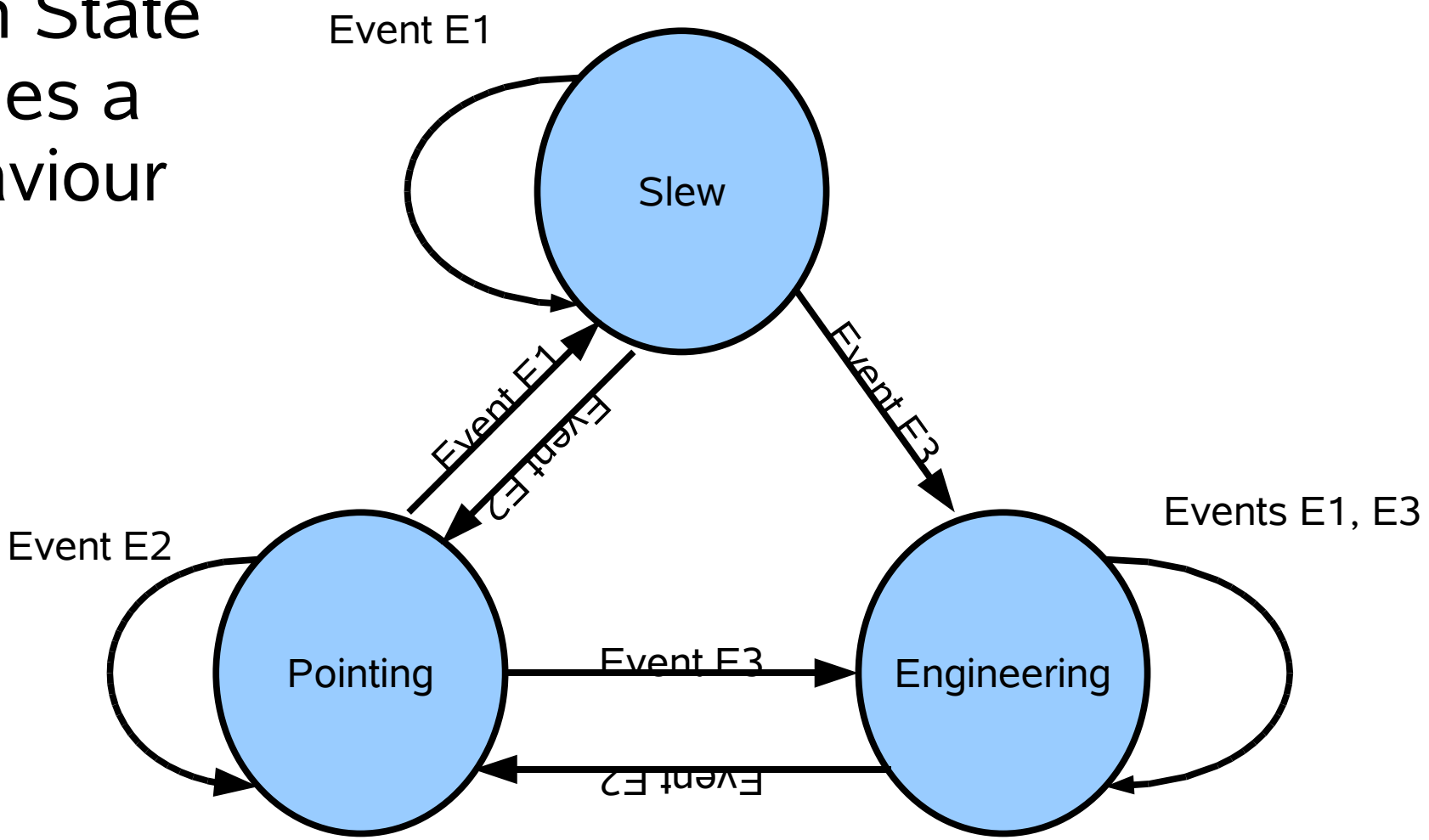

Packet
Organizer
Task Loop
(trigger)
Packet Data Parser
(Selector)
Data Driven
Parser
Code Driven
Parsers
RAW data files

Example 2

An object oriented approach to a
State Machine
representing a satellite

State Machine

Each State defines a behaviour



Traditional State Machine

- Each State is a set of variables
- Each Event is a set of variables

- The behaviour is based on the State variables
- The transitions depend on a cross combination of State and Event variables
- The calling software needs to know about states

Polymorphism based State Machine

- Each State is an object instance
- There are no State variables
- Each Event is a set of variables

- The behaviour is predefined for each State
- The transitions also depend on the Events only
- The calling software need not know the states

```
void OnNewPacket(const Packet& p)
{
if (state==Slew) {
	if (p.mode==1)
		Action1(p);
	else if (p.mode==2)
		Action2(p);
	}
else if (state==Pointing) {
	if (p.mode==1)
		Action1(p);
	else if (p.mode==2)
		Action3(p);
	}
else if (state==Engineering) {
	if (p.mode==1)
		Action2(p);
	else if (p.mode==2)
		Action1(p);
	}
}
```

```
void OnNewPacket(const Packet& p)
{
if (p.mode==1) {
	if (state==Engineering)
		Action2(p);
	else
		Action1(p);
	}
else if (p.mode==2) {
	if (state==Slew)
		Action2(p);
	else if (state==Pointing)
		Action3(p);
	else if (state==Engineering)
		Action1(p);
	}
}
```

```
void Slew::OnNewPacket(const Packet& p)
{
if (p.mode==1)
	Action1(p);
else if (p.mode==2)
	Action2(p);
}

void Pointing::OnNewPacket(const Packet& p)
{
if (p.mode==1)
	Action1(p);
else if (p.mode==2)
	Action3(p);
}

void Engineering::OnNewPacket(const Packet& p)
{
if (p.mode==1)
	Action2(p);
else if (p.mode==2)
	Action1(p);
}
```

```
class SatelliteState
{
public:
	virtual void OnNewPacket(const Packet& p) = 0;
	virtual SatelliteState* Transition(const Packet& p) = 0;
};


class Slew: public SatelliteState
{
public:
	void OnNewPacket(const Packet& p);
	SatelliteState* Transition(const Packet& p);
};
```

The top level packet handling procedure does not know the states:

```
SatelliteState* currentState;

…
currentState = currentState->Transition(p);
currentState->OnNewPacket(p);
```