

SP-ICE-3 and RAYGUIDE

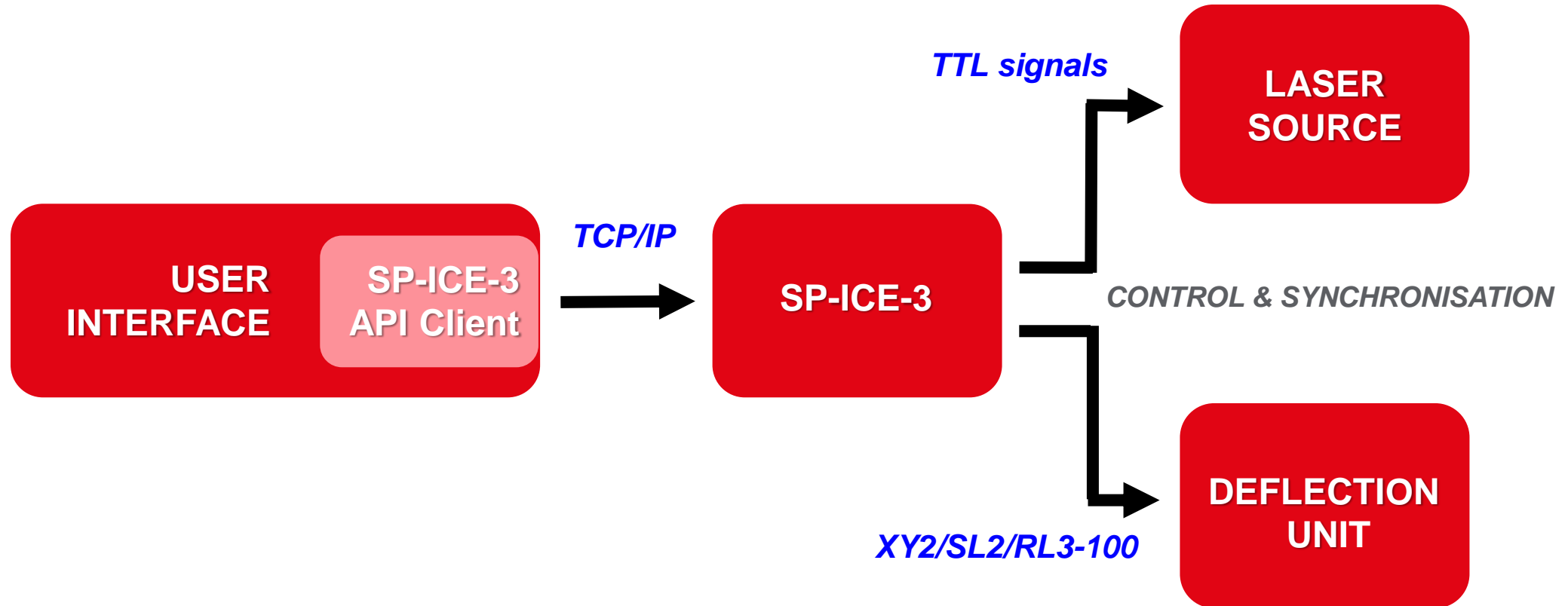
Notes on use and integration

Technical Competence Center (TCC)

Wessling 2024

LASER SCAN SYSTEM

Software & Control



SOFTWARE & CONTROL ECOSYSTEM

Layered paradigm

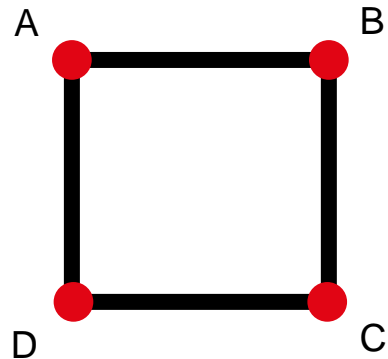


SOFTWARE ARCHITECTURAL LEVELS

Trivial exemplification

High level

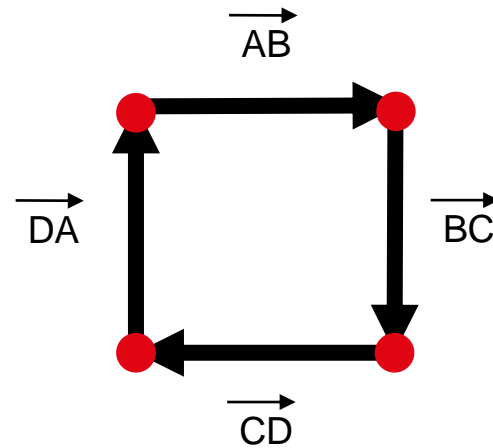
MARKING SW LIBRARY



OBJECT

Low level

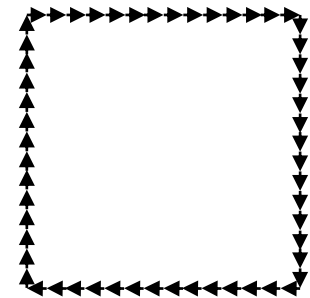
CARD LIBRARY



MACRO-VECTORS

Control card -internal level

CONTROL CARD



MICRO-VECTORS

MARKING SOFTWARE VS CARD LIBRARY

Example of creating and marking a rectangle

MARKING SW LIBRARY

```
private static MarkableRectangle CreateRectangle( double size )
{
    MarkableRectangle rectangle = new MarkableRectangle { PluginMetadata = { ShortLabel = "Rectangle" },
    Size = new dvec2( size ) };
    rectangle.MarkableConfiguration.MarkerProfile.DefaultPen.LaserPower = 1;
    rectangle.MarkableConfiguration.MarkerProfile.DefaultPen.JumpSpeed = 5.0;
    return rectangle;
}

IJobManager jobManager = marker.JobManager;

JobDefinition job = jobManager.CreateNewJob( "Mark jobElements" );
job.ScanControllers.Add( primaryScanController );
MarkableRectangle rectangle1 = CreateRectangle( fieldSize.x / 4.2 );
rectangle1.MoveTo( new dvec3( -fieldSize.x / 4, +fieldSize.y / 4, 0 ) );
job.AddJobElement( rectangle );
```

CARD LIBRARY

```
client.Laser.ArmLaser(true);
CommandList list1 = new CommandList();
list1.AppendJumpSpeed(5);
list1.AppendMarkSpeed(1);
list1.AppendJumpDelay(0);
list1.AppendMarkDelay(0);
list1.AppendPolyDelay(50);
list1.AppendLaserOnDelay(250);
list1.AppendLaserOffDelay(300);
list1.AppendLmWidth(0.5);
list1.AppendLmFrequency(2);
list1.AppendPower(65535);
list1.AppendMarkAbs(25000, 25000, 0);
list1.AppendJumpAbs(25000, -25000, 0);
list1.AppendMarkAbs(-25000, 25000, 0);
list1.AppendMarkAbs(-25000, -25000, 0);
list1.AppendMarkAbs(25000, -25000, 0);
client.List.Set(0, list1);

client.List.Execute(0);
int? id;
client.List.WaitForListDone(out id);
client.Laser.ArmLaser(false);
```

MARKING SOFTWARE VS CARD LIBRARY

MARKING SW LIBRARY

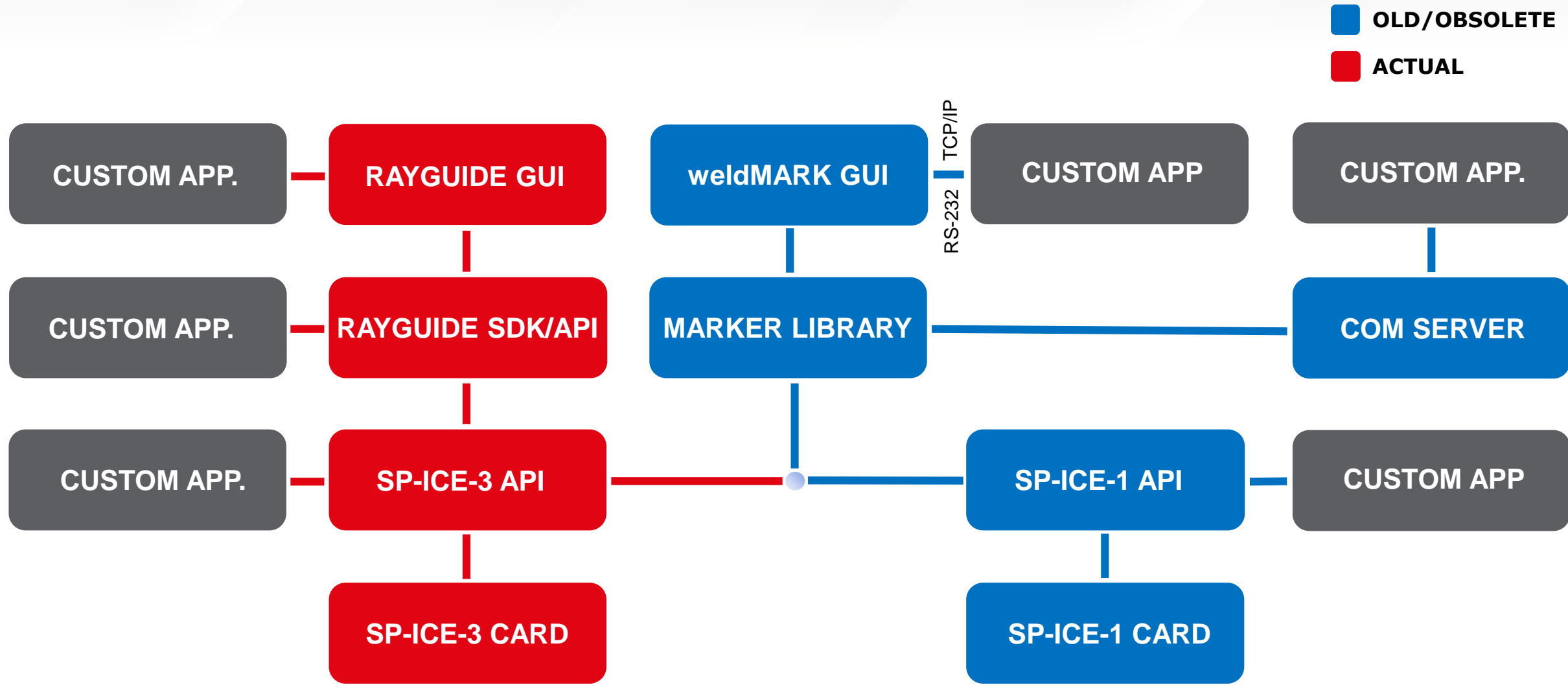
```
JobDefinition loadedJob = marker.JobManager.LoadJob("myJob.rg").JobDefinition;  
loadedJob.ScanControllers.Add(primaryScanController);  
marker.JobManager.RunSync(loadedJob, typeof(OnHostJobExecutor));
```

* With the use of the marking library, one can not only create jobs programmatically, but also simply load GUI-created jobs, manipulate and execute them with ease.

CARD LIBRARY

```
client.Laser.ArmLaser(true);  
CommandList list1 = new CommandList();  
list1.AppendJumpSpeed(5);  
list1.AppendMarkSpeed(1);  
list1.AppendJumpDelay(0);  
list1.AppendMarkDelay(0);  
list1.AppendPolyDelay(50);  
list1.AppendLaserOnDelay(250);  
list1.AppendLaserOffDelay(300);  
list1.AppendLmWidth(0.5);  
list1.AppendLmFrequency(2);  
list1.AppendPower(65535);  
list1.AppendMarkAbs(25000, 25000, 0);  
list1.AppendJumpAbs(25000, -25000, 0);  
list1.AppendMarkAbs(-25000, 25000, 0);  
list1.AppendMarkAbs(-25000, -25000, 0);  
list1.AppendMarkAbs(25000, -25000, 0);  
client.List.Set(0, list1);  
  
client.List.Execute(0);  
int? id;  
client.List.WaitForListDone(out id);  
client.Laser.ArmLaser(false);
```

RAYLASE-SPECIFIC ECOSYSTEM



RAYGUIDE

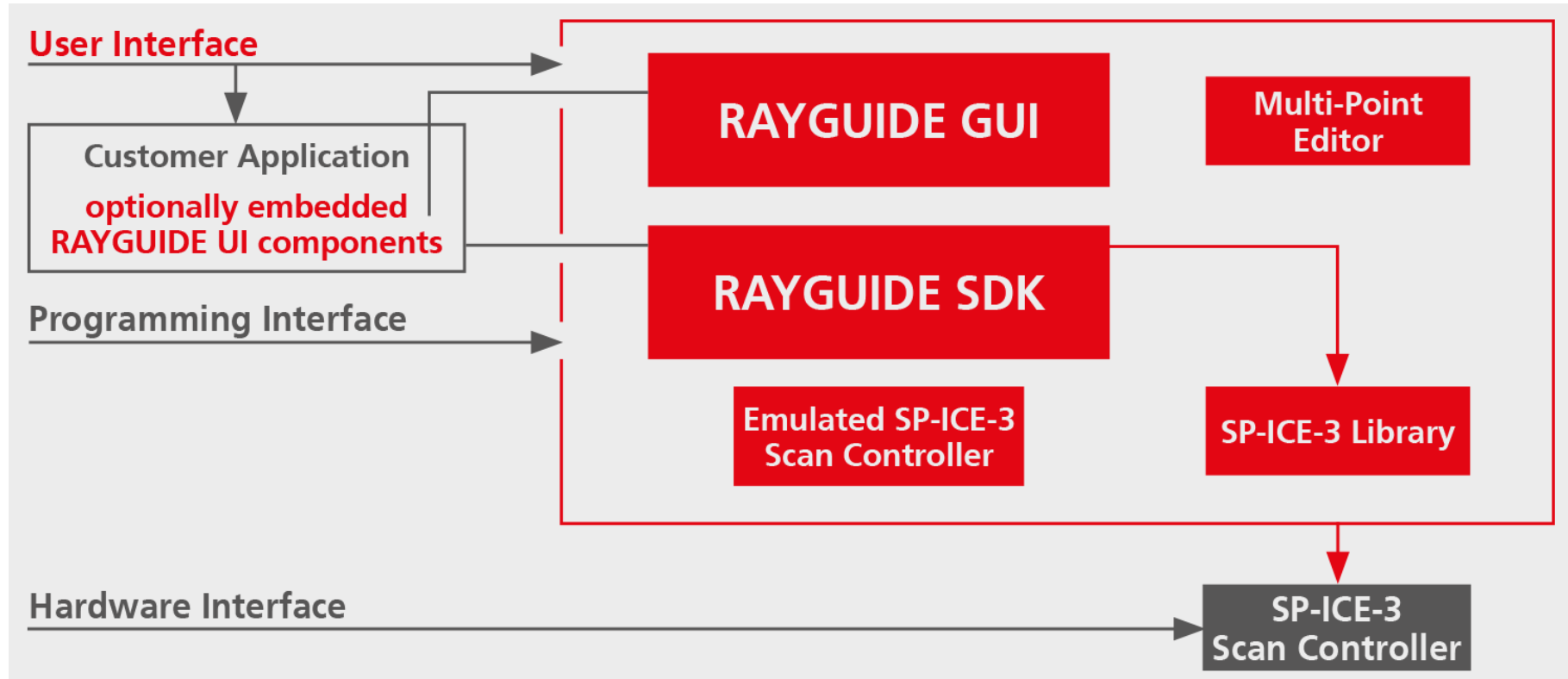
General description

RAYGUIDE is a laser processing software suite. Its powerful and flexible UI makes it easy to design or import text, barcodes, and graphic elements in order to realize complex and extensive laser process projects. The RAYGUIDE suite offers two main user interfaces:

- The **RAYGUIDE Graphical User Interface** (GUI), which allows users to directly manage complex laser processing jobs without programming knowledge.
- The **RAYGUIDE Software Development Kit** (SDK), a programmable Application Programming Interface (API), based on the Microsoft .NET environment. This means that the entire functionality of RAYGUIDE can be integrated into a customized system application entirely according to customer requirements.

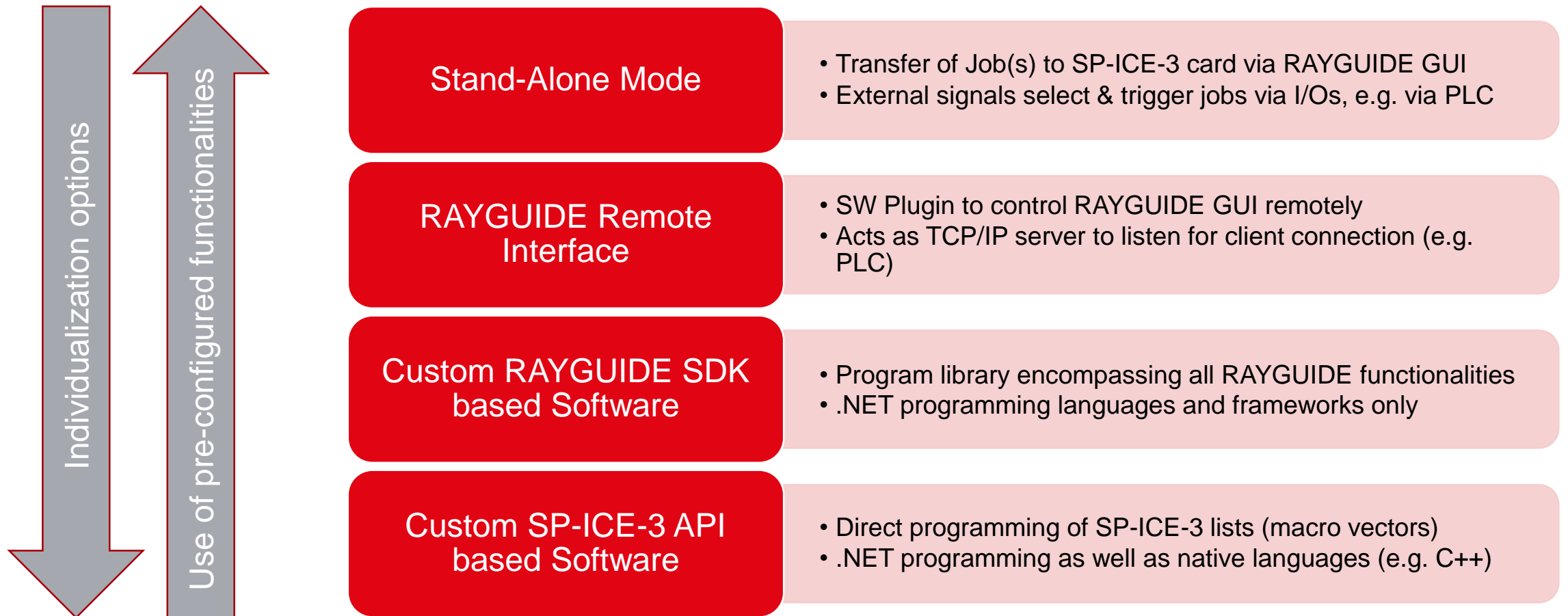
RAYGUIDE ECOSYSTEM

Schematic



INTEGRATION OPTIONS

Overview



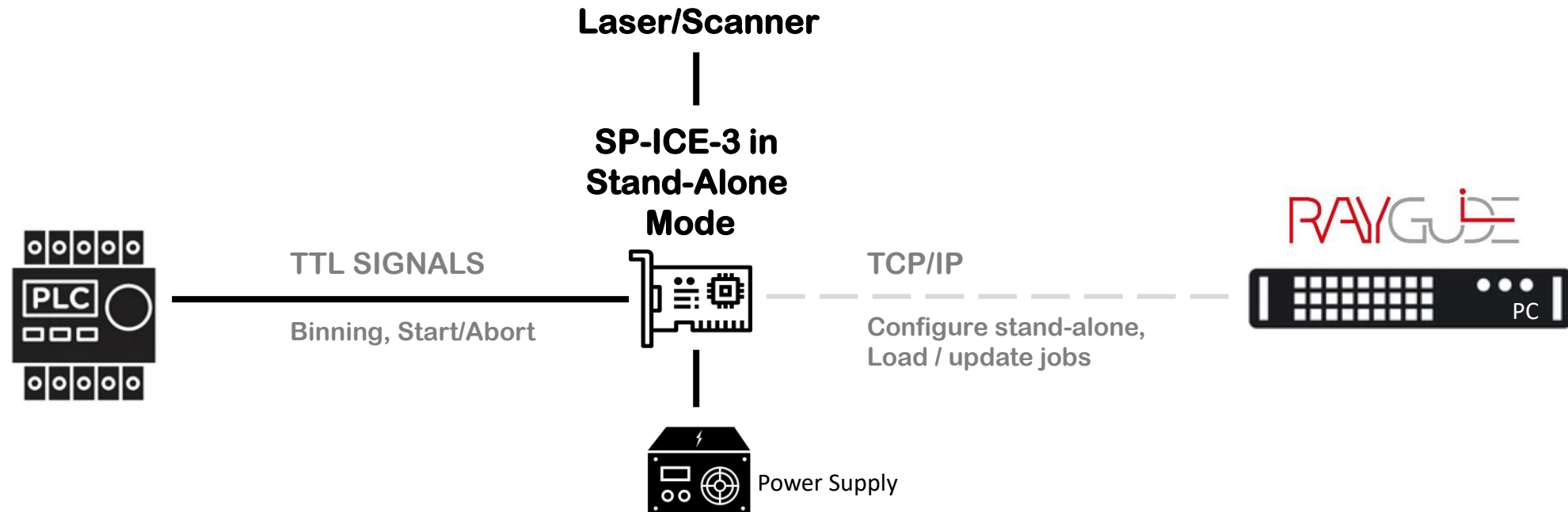
STAND ALONE MODE

General description

- The stand-alone operation mode of the SP-ICE-3 scan controller card enables the preconditioned automatic and autonomous execution of RAYGUIDE jobs.
- The creation of the jobs as well as the stand-alone mode configuration takes place in RAYGUIDE GUI. At this stage the RAYGUIDE GUI needs to be connected to the scan controller.
- You can transfer one or more jobs to the scan controller memory; these will remain stored on the card even if the scan controller is rebooted or powered off.
- Moreover you can define I/O conditions (binning) for the external selection and execution triggering or aborting of these jobs.
- Thereafter the card can execute the jobs independently i.e. without being connected to the RAYGUIDE GUI or being monitored by it.
- During machine operation, external TLL signals are sent to the scan controller and received via the input I/Os for job selection as well as start/stop processing. Typically, these signals are generated by a PLC.

STAND ALONE MODE

Schematic



* *Binning*

Using signal patterns from external devices, for example a PLC, can be used to control whether a job (stand-alone mode only) or an object should be processed.

— permanent connection
- - - temporary connection

STAND ALONE MODE

Advantages and Limitations

Advantages	Limitations
Quick/simple integration	No in-process parameter adjustment possible
Faster response times	Objects with dynamic content not possible
No PC required in the operational machine	Very primitive error notifications – no granularity
Typical PLC programming skills sufficient	
Only one RAYGUIDE license required for the commissioning cards with the stand-alone mode; no RAYGUIDE license required for each machine running on stand-alone	

STAND ALONE MODE

Target Group

- Customers with typical automation i.e. PLC programming capacity
- Customers with no capacity to develop their own software based on RAYGUIDE SDK or SP-ICE-3 API
- Customers with applications requiring only a finite number of jobs for relatively long periods of machine operation
- Customers with applications that do not require in-process parameter changes (e.g. power or position adjustments at each execution cycle or otherwise regularly during machine operation)

* (Stand-alone mode also configurable i.e. programmable per RAYGUIDE SDK and SP-ICE-3 API)

RAYGUIDE REMOTE INTERFACE

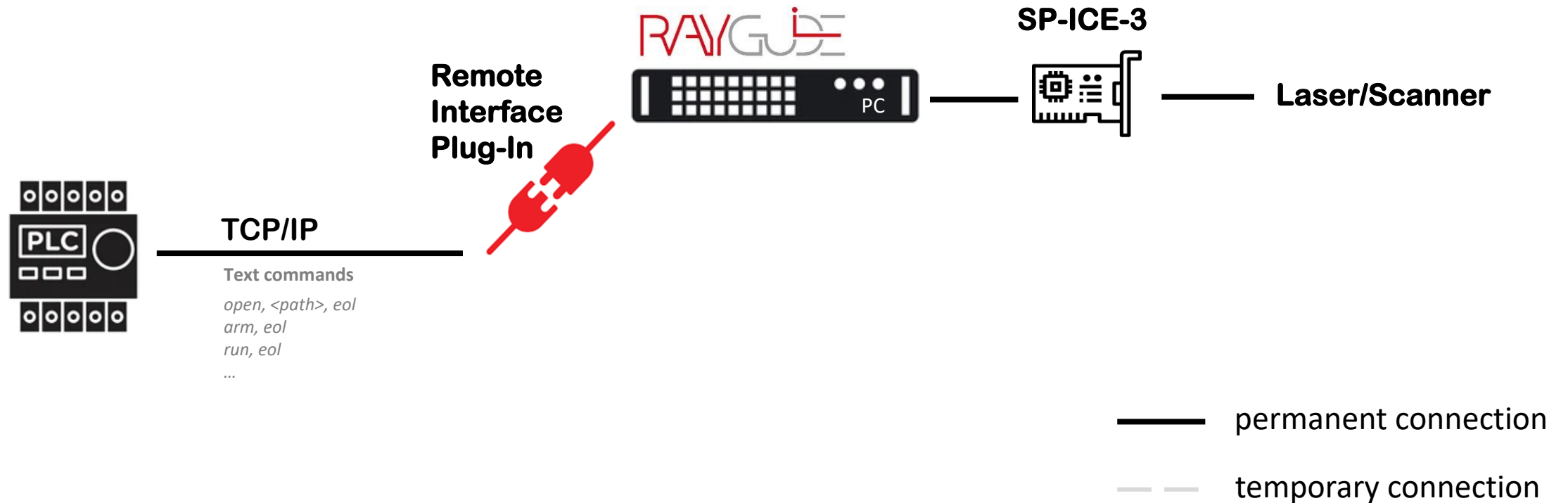
General description

- RAYGUIDE GUI plugin that enables its remote controllability
- The Remote Interface plugin acts as a TCP/IP server set to listen for a TCP/IP client connection
- The IP address of the PC that hosts RAYGUIDE GUI is used for this connection (not the IP of the SP-ICE-3 card! This is used for the connection between RAYGUIDE GUI and the SP-ICE-3)
- After connection, the TCP/IP client can send commands as text strings and thus control the RAYGUIDE GUI remotely. These commands typically encompass not only loading and triggering jobs, but also modifying process parameters and job objects' positions.

** a plug-in or plugin, is a software component that adds a specific feature to an existing computer program. When a program supports plug-ins, it enables customization.*

RAYGUIDE REMOTE INTERFACE

Schematics



RAYGUIDE REMOTE INTERFACE

Advantages and Limitations

Advantages	Limitations
In-process parameter updates possible to a significant extent	More complicated integration
Objects with dynamic content possible	Slower and more stochastic response times (due to nature of TCP/IP communication)
More granular error notifications	PC required in the machine (for RAYGUIDE GUI to run)
Typical PLC programming skills sufficient	RAYGUIDE license required at the/each machine's PC

RAYGUIDE REMOTE INTERFACE

Target Group

- Customers with typical automation i.e. PLC programming capacity
- Customers with no capacity to develop their own software based on RAYGUIDE SDK or SP-ICE-3 API
- Customers with applications requiring an infinite number of jobs that may change over machine operation time more frequently
- Customers with applications that require in-process parameter changes (e.g. power or position adjustments at each cycle)

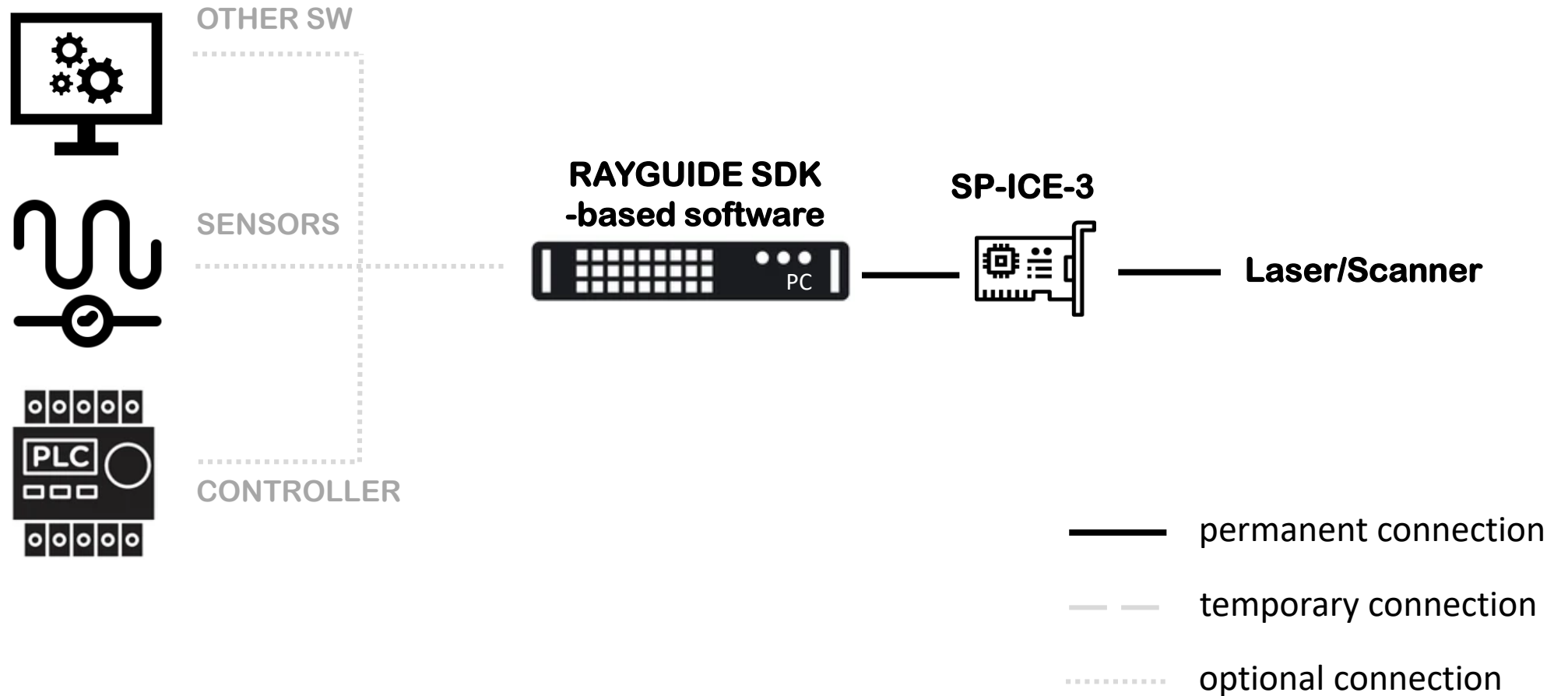
RAYGUIDE SDK

General description

- The RAYGUIDE marking software suite is implemented with the idea of easy reusability of its functionalities for developers. Our approach is to provide the developer with an Application Programming Interface (API) which is also used by the RAYGUIDE GUI.
- The RAYGUIDE SDK (marking software application building blocks) would be primarily of interest for developers who want to integrate RAYGUIDE's functionality into their own applications.
- With the use of the RAYGUIDE SDK one cannot only use the RAYGUIDE APIs functions, but even embed parts of RAYGUIDE GUI into its own application.
- Since RAYGUIDE and its API is based on Microsoft's .NET framework and implemented in C#, knowledge of this environment will be needed and is presupposed here. The RAYGUIDE SDK is available for the .NET environment only.

RAYGUIDE SDK

Schematic



RAYGUIDE SDK

Advantages and Limitations

Advantages	Limitations
Full flexibility of all RAYGUIDE functions	.NET programming skills required
Implement only the existing functions you need	Possibly difficult for developers that have no previous laser marking software experience
Create your own functions	Not available for programming languages outside the .NET framework
Individualization of GUI - no RAYLASE branding possible	
Effortless import & optimization of CAD files	
Image acquisition & processing functions available	
Math libraries to handle easily matrices, vectors & transformations	

RAYGUIDE SDK

Target Group

- Customers with .NET programming capacity
- Customers who aim at creating their own tailored-made software, based on the RAYGUIDE marking software's building blocks, which already encompass higher level logic e.g. functionalities such as import of CAD drawings, automation objects, profiles, etc.
- Customers that are already familiar with the RAYGUIDE GUI
- Customers who can tolerate intermediate to longer development timelines

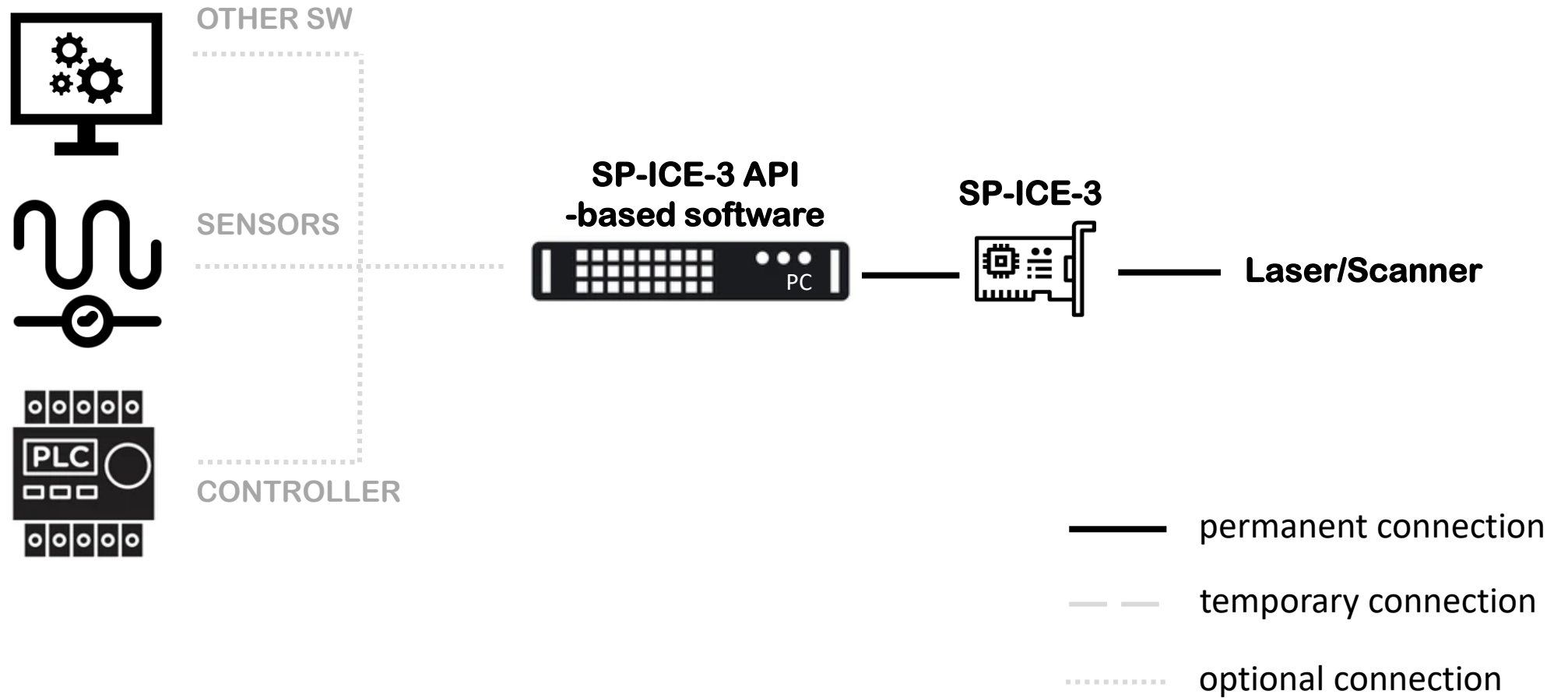
SP-ICE-3-API

General description

- The SP-ICE-3 control card is the universal solution for every laser system with deflection units. It comes with its own, low-level Application Programming Interface (API), independent of the higher-level RAYGUIDE SDK/API. The main purpose of the SP-ICE-3 API is to abstract communications directly with the SP-ICE-3 Card.
- Thanks to the managed .NET and native C(++) libraries and the flexibility of the SP-ICE-3 control card, a broad variety of applications can be solved with the highest level of flexibility according to one's specific requirements.
- By using the SP-ICE-3 API one can perform all necessary configurations and most importantly create Command Lists for execution. Fundamentally, a Command List is a container for SP-ICE-3 commands and their Parameters; these commands are generally aimed at specifying positions for the laser beam to visit (mark & jump vectors), interspersed with other commands which affect or effect laser timing, laser power levels, I/O signals, and flow-control.

SP-ICE-3 API

Schematic



SP-ICE-3-API

Advantages and Limitations

Advantages	Limitations
Maximum flexibility & individualization	.NET or native (e.g. C++) programming skills required
Create your own high-level functions	Shallow learning curve for developers that have no previous laser marking software experience
Seamless processing between lists, no dead-time due to communication overhead (except maybe TTL signals)	No high order functionalities such as CAD files import, Barcodes or Texts
process transformation / dynamic content	TCP-IP not real-time capable, long reaction time for setting of process transformation

SP-ICE-3-API

Target Group

- Customers that have the capacity to develop custom software using managed i.e. .NET as well as native programming languages (e.g. C++); other programming languages such as Python also possible, but are not actively supported.
- Customers that want to develop their own custom-tailored higher level logic to meet their exact requirements without adopting the RAYGUIDE marking software suite.
- Customers who can tolerate intermediate to longer development timelines

DECISION CHART

KEY QUESTIONS FOR DECISION MAKING:


1. Programming capacity? Managed (.NET) or Native (C++)?
2. Development timeline?
3. RAYLASE or own-branded UI?
4. Type of application? (AM, welding (which kind?), cutting/perforating, etc.)
5. In-process parameter updates?
6. Reaction/response time requirements?


Thank You

Get connected for the POWER of WE

Dr.-Ing. Jan Bernd Habedank

Head of Technical Competence Center

 j.habedank@raylase.de

 +49.8153.99.99.233